IBM

RS/6000 SP System Performance Tuning Update

What you always wanted to know about SP performance but were afraid to ask

New tuning information about PSSP 3.2 and AIX 4.3.3

Sample tuning scripts included

Dino Quintero
Jess Hart
Maria Lafranke
Bjorn Roden
Theeraphong Thitayanun
Marc-Eric Kahle

Redbooks

ibm.com/redbooks

SG24-5340-01

International Technical Support Organization

RS/6000 SP System Performance Tuning Update

January 2001

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, “Special notices” on page 487.

Second Edition (January 2001)

This edition applies to Version 3, Release 2 of the IBM Parallel System Support Programs for AIX (PSSP) for use with AIX Operating System Version 4 Release 3 (5765-C34).

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Building 003 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

Contents

Figures xi

Tables xiii

Preface xv
The team that wrote this redbook xv
Comments welcome xvii

Book layout xix

Start map xxi

Part 1. Overview 1

Chapter 1. Introduction 3
1.1 Why tune? 3
1.2 When to tune? 3
1.3 Why SP tuning differs from normal AIX tuning? 3

Chapter 2. Tuning methodology 5
2.1 The SP performance tuning cycle 5

Part 2. General tuning 11

Chapter 3. Basic performance settings 13
3.1 Basic network settings 14
3.2 Basic adapter settings 15
3.3 Basic AIX system settings 18

Chapter 4. Network tuning 21
4.1 Initial considerations 21
4.1.1 General tuning recommendations 21
4.1.2 Consolidated system challenges 22
4.1.3 System topology considerations 22
4.2 TCP/IP network overview for performance 23
4.2.1 Send and receive flows in the TCP/IP protocol stack 24
4.2.2 MTU 28
4.2.3 Communication subsystem memory (mbuf) management 29
4.2.4 Enabling thread usage on LAN adapters (dog threads) 32
4.2.5 Tuning TCP maximum segment size 33
4.2.6 TCP sliding window 36

4.2.7 Address resolution 40
4.2.8 Subnet addressing 41
4.2.9 Description of the Nagle Algorithm 42
4.2.10 Routing. 45
4.3 AIX network tunables 45
4.3.1 IP tunable inheritance rules 45
4.3.2 AIX tunables. 46
4.4 SP system-specific network tunables 67
4.5 NFS tuning 68
4.5.1 NFS overview for performance 69
4.5.2 Read/write throughput 70
4.5.3 Tuning the numbers of nfsd and biod daemons 72
4.5.4 Tuning to avoid retransmits 73
4.5.5 Dropped packets 75
4.5.6 Cache file system 78
4.5.7 Reduced requests for file attributes 78
4.5.8 Disabling unused NFS ACL support 79
4.5.9 NFS tuning checklist. 79
4.5.10 Examples of NFS configurations. 82
4.6 SP system-specific tuning recommendations 83
4.6.1 IBM-supplied files of SP tunables 85
4.7 Tuning the SP network for specific workloads. 86
4.7.1 Tuning for development environments 86
4.7.2 Tuning for scientific and technical environments. 87
4.7.3 Tuning for commercial and database environments 89
4.7.4 Tuning for server environments 91
4.7.5 Summary of workload tunables. 93

Chapter 5. Adapter tuning 95
5.1 Adapter queue size 95
5.1.1 Transmit and receive queues 96
5.1.2 Displaying adapter queue settings 98
5.1.3 Changing adapter settings 98
5.1.4 Adapter tuning recommendations 99
5.2 SP Switch and SP Switch2 adapter tuning 99
5.2.1 Switch adapter pools 99
5.2.2 Window sizes 100
5.2.3 Switch window reservation 103
5.2.4 Switch pool allocation 105
5.2.5 Switch buffer pool allocation considerations 106
5.2.6 Sizing send and receive pool requirements. 107
5.2.7 Sample switch send pool size estimate. 109
5.2.8 Reducing send/receive pool requirements 110

5.3 SP Ethernet tuning 110
5.4 Gigabit ethernet performance tuning recommendations 111
5.5 Token-Ring performance tuning recommendations 112
5.6 FDDI performance tuning recommendations 112
5.7 ATM performance tuning recommendations 112
5.8 HIPPI performance tuning recommendations 112
5.9 Escon interface tuning 113

Chapter 6. Tuning other components in AIX 115
6.1 Memory 115
6.1.1 Virtual memory manager 116
6.1.2 Real-memory management 117
6.2 CPU 129
6.2.1 Scheduler run queue management 129
6.2.2 CPU tuning 136
6.3 I/O 141
6.3.1 Logical volume manager 141
6.3.2 Data transfer path to and from disk 142
6.3.3 Disk types 144
6.3.4 Improving read and write performance (VMM) 152
6.3.5 Logical Volume Device Driver 155
6.3.6 Performance implications of disk mirroring 157
6.3.7 Logical volume striping 161
6.3.8 JFS and JFS log 164
6.3.9 Paging space allocation 166
6.3.10 Tuning I/O pacing 167
6.3.11 Asynchronous I/O 169
6.3.12 Logical volume policies 174
6.3.13 File system fragmentation 178

Chapter 7. Control workstation tuning 181
7.1 Control workstation considerations 181
7.2 Control workstation tuning 182
7.2.1 Network tunables 182
7.2.2 Adapter tuning 183
7.2.3 Other AIX components tuning 183
7.2.4 Other considerations 183

Part 3. Problem determination and tools 187

Chapter 8. Problem determination 189
8.1 Performance problem checklist 189

Chapter 9. Common performance problems . . . 191
9.1 The Nagle Algorithm . . . 191
9.2 Adapter queue overflows . . . 192
9.3 ARP cache tuning . . . 193
9.4 NFS performance in SP system . . . 194
9.5 External server considerations . . . 195
9.6 Single-server multiple-client node problems . . . 197
9.7 Gateway or router node problems . . . 200
9.8 Typical performance problems by environments . . . 200

Chapter 10. Resource monitoring . . . 203
10.1 IBM AIX monitoring tools and commands . . . 203
10.2 Monitoring memory . . . 204
10.2.1 System-wide monitoring . . . 205
10.2.2 Application specific monitoring . . . 208
10.2.3 User specific monitoring . . . 211
10.3 Monitoring CPU . . . 212
10.3.1 System-wide monitoring . . . 213
10.3.2 Application specific monitoring . . . 218
10.4 Monitoring disk I/O . . . 222
10.4.1 System-wide monitoring . . . 222
10.5 Monitoring Network I/O . . . 231
10.5.1 System-wide monitoring . . . 232

Chapter 11. Tools . . . 243
11.1 IBM AIX Monitoring tools and commands . . . 243
11.1.1 acctcom . . . 243
11.1.2 bf/bfrpt . . . 244
11.1.3 crash . . . 246
11.1.4 col_dump . . . 247
11.1.5 entstat/tokstat/atmstat/fddistat . . . 248
11.1.6 estat . . . 251
11.1.7 filemon . . . 251
11.1.8 fileplace . . . 256
11.1.9 hatstune . . . 258
11.1.10 ifcl_dump . . . 260
11.1.11 iostat . . . 260
11.1.12 iptrace/ipreport/ipfilter . . . 262
11.1.13 lsps . . . 265
11.1.14 lspv/lslv/lsvg/getlvcb . . . 266
11.1.15 mmlsmgr/mmdf/mmlsfs/mmlsdisk/mmfsadm . . . 269
11.1.16 ndb . . . 271
11.1.17 netpmon . . . 275

11.1.18 netstat 281
11.1.19 nfsstat 287
11.1.20 nslookup/host 293
11.1.21 ntpq 295
11.1.22 ntptrace 295
11.1.23 Performance Diagnostic Tool (PDT) 295
11.1.24 ping 302
11.1.25 pprof 303
11.1.26 ps 304
11.1.27 pstat 308
11.1.28 sar 309
11.1.29 spmon 313
11.1.30 statvsd 313
11.1.31 svmon 314
11.1.32 tcpdump 317
11.1.33 topas 319
11.1.34 tprof 320
11.1.35 traceroute 321
11.1.36 trace, trcrpt 322
11.1.37 vmstat 324
11.1.38 vmtune 330
11.1.39 xmperf, 3dmon 331

Part 4. Tuning selected products 335

Chapter 12. Tivoli Storage Manager (ADSM) 337
12.1 TSM tunables - quick reference 337
12.2 Performance considerations 338
12.3 SP network tunables 339
12.4 Server performance tuning 339
12.4.1 Server Options to improve network performance 340
12.4.2 Server options to improve memory performance 342
12.4.3 Server options to improve CPU performance 343
12.4.4 Server options to improve I/O performance 344
12.4.5 Server options to improve TSM server database performance 345
12.5 Client performance tuning 345
12.5.1 Client options to improve network performance 346
12.5.2 Client options to improve memory performance 346
12.5.3 Client options to improve CPU performance 347
12.5.4 Client options to improve I/O performance 348
12.5.5 Client options to improve small file size performance 348
12.6 Some test results 348

Chapter 13. VSD . . . 353
13.1 Overview . . . 353
13.2 Tuning . . . 354
13.2.1 Application considerations . . . 354
13.2.2 I/O subsystem considerations . . . 355
13.2.3 SP switch considerations . . . 355
13.2.4 VSD parameters: . . . 356
13.2.5 Summary of VSD tuning recommendations . . . 362
13.3 Other VSD considerations . . . 363
13.3.1 Concurrent VSD . . . 363
13.3.2 VSD usage of KLAPI . . . 364

Chapter 14. GPFS . . . 369
14.1 Overview . . . 369
14.2 Tuning . . . 373
14.2.1 Network tuning . . . 373
14.2.2 Switch adapter tuning . . . 374
14.2.3 SSA adapter tuning . . . 375
14.2.4 VSD tuning . . . 377
14.2.5 GPFS tuning . . . 380
14.3 Summary of GPFS tuning recommendations . . . 390

Chapter 15. Web applications . . . 393
15.1 Tuning web applications on the SP . . . 393
15.1.1 Websphere - an overview . . . 393
15.1.2 What can be tuned? . . . 394
15.1.3 AIX tunables for web related applications . . . 395
15.1.4 Performance test tools for webserver . . . 396
15.1.5 Monitoring tools . . . 398

Chapter 16. DB2 Universal Database . . . 401
16.1 General concepts about parallel databases . . . 402
16.1.1 Inter-partition and intra-partition parallelism . . . 403
16.1.2 Database partitioning . . . 404
16.2 Tuning database systems for performance . . . 405
16.2.1 Document the current system . . . 405
16.2.2 Bottlenecks, utilization, and resources . . . 407
16.2.3 Insufficient CPU and latent demand . . . 408
16.2.4 Insufficient memory . . . 408
16.2.5 Insufficient network resources . . . 410
16.2.6 Insufficient logical resource access . . . 410
16.2.7 What can we tune? . . . 411
16.2.8 Top 20 DB2 parameters . . . 412
16.3 DB2 UDB Implementation on RS/6000 SP: a real case . . . 420

16.3.1 Hardware configuration 420
16.3.2 Software configuration 422
16.3.3 Database layout 422
16.3.4 Operating system tuning 424
16.3.5 Database tuning 425
16.3.6 System management and performance 425
16.3.7 Additional tuning observations 430

Chapter 17. Oracle 433
17.1 Oracle Parallel Server (OPS) 433
17.1.1 Parallel Oracle architecture 434
17.1.2 Virtual Shared Disk (VSD) 438
17.1.3 Distributed lock manager (DLM) 439
17.2 Oracle tuning 440
17.2.1 Oracle tuning order 441
17.2.2 Tuning AIX for Oracle 442
17.2.3 Top 10 Oracle parameters 450

Chapter 18. Lotus Notes 455
18.1 What can be tuned now in the SP and AIX environments? 455
18.2 Transactional logging and how it operates 456
18.2.1 Performance improvement 457
18.2.2 Flushing and hardening 458
18.2.3 Crash recovery 458
18.2.4 Transactional Logging NOTES.INI Parameters 459
18.3 Tuning servers for maximum performance 459
18.3.1 Server tasks 459
18.3.2 Server performance problems 460

Chapter 19. Communication server 463
19.1 Communications buffers (mbufs) 463
19.1.1 Setting maxmbuf 464
19.2 SNA DLC link parameters 464
19.2.1 LAN-based DLC characteristics 464
19.3 Memory 465
19.4 Paging space 465
19.5 File transfer applications 466

Part 5. Appendices 467

Appendix A. Performance toolbox for AIX and parallel extensions . . 469
A.1 Performance toolbox for AIX (PTX/6000) 469
A.1.1 PTX/6000 installation 470

A.2 Performance toolbox parallel extensions (PTPE) 472
A.2.1 PTPE installation 472

Appendix B. How to report performance problems to IBM 475

Appendix C. Hardware details 479
C.1 Node types 479
C.2 Roles of nodes 480
C.3 Communication paths 482
C.4 System partitioning 482
C.5 Node selection process 483

Appendix D. Special notices 487

Appendix E. Related publications 491
E.1 IBM Redbooks 491
E.2 IBM Redbooks collections. 491
E.3 Other resources 492
E.4 Referenced Web sites. 492

How to get IBM Redbooks 495
IBM Redbooks fax order form 496

Index 497

IBM Redbooks review 515

Figures

1. Performance tuning cycle 5
2. TCP/UDP/IP data flow 25
3. The network memory pool 30
4. Socket layer 31
5. Subnet addressing 35
6. Inter-subnet fragmentation 36
7. TCP Sliding Window 37
8. rfc1323 - TCP extension 38
9. Inserting no options for ARP cache in /etc/rc.net. 41
10. The Nagle Algorithm 43
11. netstat -I command 44
12. NFS overview 68
13. Adapter queue overview 95
14. MTU ratio 97
15. lsattr command output for an ATM adapter 98
16. Viewing send and receive pool buffer sizes 100
17. chgcss command 102
18. lsattr -El css0 output 103
19. Switch pool allocation 105
20. Output of the vdidl3 command 108
21. Querying fragmentation for a file system 179
22. Defragmentation of file system with defragfs 180
23. Calculating tcp send/receive space sizes 196
24. Single-server multiple-client scenario 198
25. Sample setsockopt() call 199
26. xmperf custom output for memory 211
27. 3dmon CPU monitor 218
28. xmperf disk I/O monitor 231
29. xmperf ntework I/O monitor 242
30. Monitoring a system using xmperf 332
31. xmperf 333
32. Default client and server files 349
33. Changed client dsm.sys file 350
34. Changed server dsmserv.opt file 351
35. VSD implementation 353
36. Write data flow: VSD using IP 366
37. Write data flow: VSD using KLAPI 366
38. GPFS overview 372
39. Fundamental tuning points of WebSphere application server settings . . 395
40. Resource Analyzer with case1 (-mx64m) 398

41. Shared nothing. 402
42. DB2 UDB EEE structure 404
43. Balancing paging I/O against database I/O 409
44. The SSA disk layout. 422
45. Logical volume layout. 423
46. CPU utilization. TPC-H power run. 427
47. I/O characteristics. TPC-H power run 428
48. CPU utilization. TPC-H throughput run. 429
49. I/O bandwidth. TPC-H throughput run. 430
50. An Oracle instance. 435
51. General Oracle Parallel Server architecture 436
52. Parallel Oracle on SP systems 437
53. Oracle tables working with VSD operations. 438
54. Distributed lock manager operation 439
55. SGA buffer cache memory sizes for raw device and JFS DBs 451
56. PTX/6000 network monitoring - client/server model 469
57. RS/6000 SP node selection based on capacity 484
58. RS/6000 SP node selection based on performance 485

Tables

1. Start map xxi
2. Basic network settings with the no command 14
3. Maximum transmission units 29
4. Default ARP parameters in AIX 40
5. TCP/IP Pacing Degradation Window 44
6. Socket level network option tunables. 49
7. Recommended values for ARP cache sizing in SP systems 58
8. Software development tuning parameters 86
9. Scientific and technical environment tuning parameters 88
10. Commercial and database environment tuning parameters 90
11. Server tuning parameters 92
12. Summary of workload tunables 93
13. Transmit queue size examples 96
14. Window types. 101
15. settings for new attributes 102
16. vdidl3xx commands 108
17. Escon interface tuning parameters 113
18. Backing store for different memory segment types 118
19. Simplified transfer path of data from disk to memory. 142
20. Terms used in describing disk device block operations. 143
21. Latencies for disk access times 143
22. Implications on performance of mirror write scheduling policies 160
23. Sample values for minpout and maxpout. 169
24. CWS network tunables 182
25. Settings for nodes running primarily parallel jobs 185
26. Settings for nodes running primarily DB or mixed loads 185
27. Settings for nodes with intensive I/O and heavy paging 185
28. Determining ARP tuning settings based on the number of nodes 193
29. Determining ARP tuning settings based on number of IP interfaces . . . 194
30. Common performance problems categorized by environment 200
31. IBM AIX monitoring tools by system resources 203
32. Commands for determining network protocol and services settings 231
33. Header types 264
34. Files produced by pprof 303
35. TSM server values in dsmserv.opt. 337
36. TSM client values in dsm.sys. 338
37. no parameter values 339
38. BufPoolSize recommended sizes 342
39. Parameters that are now internally controlled by GPFS 390
40. Summary for tuning a GPFS subsystem 390

41. Tunable no command parameters 396
42. Bottleneck thresholds depend on the resource and workload 407
43. System parameters tuned 424
44. DB2 UDB parameters tuned 425
45. Variables used by ptxrlog. 426
46. Memory distribution for JFS and raw device databases 444
47. Server tasks that may be disabled. 460
48. Checklist 475
49. Installation of perfpmr 477
50. Usage of perfpmr 478
51. SP nodes overview 479

Preface

This IBM redbook provides an updated version of the SP System Performance Tuning Guide. It now includes new information about changes in AIX V4.3.3 and PSSP V3.2 that can affect your SP System performance. Also, new hardware that is available especially for the SP, such as the new SP Switch2, is described along with its enhancements and tunables.

This redbook now includes information about certain applications: Tivoli Storage Manager (ADSM), DB2, and GPFS, just to name a few. To make this book comprehensive, we added more specific information about monitoring your system to increase performance. This also includes some useful scripts that can help you improve system performance or analyze the current settings of your system faster and easier.

The reader of this book can range from a novice of performance tuning to an experienced administrator. Considerable thought has gone into the layout of the book. Readers with greater experience in SP tuning have new “quick reference” features, such as startmaps and part guides: this will make referencing parameter and other environment information easier. Full and clear explanation of concepts and procedures are still provided.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Dino Quintero is a project leader at the International Technical Support Organization (ITSO), Poughkeepsie Center. He has over nine years of experience in the Information Technology field. He holds a BS in Computer Science, and a MS degree in Computer Science from Marist College. Before joining the ITSO, he worked as a Performance Analyst for the Enterprise Systems Group, and as a Disaster Recovery Architect for IBM Global Services. He has been with IBM since 1996. His areas of expertise include Enterprise Backup and Recovery, Disaster Recovery Planning and Implementation, and RS/6000. He is also a Microsoft Certified Systems Engineer. Currently, he focuses on RS/6000 Cluster technology by writing redbooks and teaching IBM classes worldwide.

Jess Hart is a SP Systems Administrator at IBM Hursley Park in the UK. She has ten years experience in the AIX field. She holds a degree in Industrial Economics from Coventry Polytechnic. She previously worked for IBM in AIX technical support, specializing in SP and SMP systems, and has written and presented workshops in both these areas. She is a Certified Advanced Technical Specialist for AIX and SP.

Marc-Eric Kahle is a RS/6000 Hardware Support specialist at the IBM ITS Central Region SPOC in Germany. He has eight years of experience in the RS/6000 and AIX fields. He has worked at IBM Germany for 13 years. His areas of expertise include RS/6000 Hardware, including the SP, and he is also certified in AIX. A lot of his experience with the SP System comes from several lab projects in Poughkeepsie, onsite support at some of the biggest European SP customers and the so-called Top Gun specialist workshops for SP in Poughkeepsie.

Maria Lafranke is an IT Specialist in IBM ITS Division in Spain. She has worked at IBM Spain for five years in all areas concerning AIX and SP. She holds a degree in Computer Science from Polytechnic University in Madrid. Also, she is a Certified Advanced Technical Expert in AIX and SP.

Bjorn Roden was recruited by IBM in the late 1980s while he was working as a programmer, and studied for the last year at Lund University in Sweden for his Master of Science. After working three years as SE (only with AIX) for IBM in Malmo, and during the last year as the person responsible for local AIX competence, he started a RS/6000 Business Partner with a former IBM colleague, which became the second largest RS/6000 Business Partner in Sweden. During his time at UNIMAX, Bjorn has achieved all SP, HACMP, ADSM and AIX Advanced Technical Expert certifications. He has also, during this time, been in charge of installing the largest SP installations for industrial customers in Sweden. Bjorn now works as the technical manager for Sweden’s largest RS/6000 Business Partner, Pulsen Systems.

Theeraphong Thitayanun is a Consulting IT Specialist with IBM Thailand. He has been with IBM for 12 years. His main responsibility is to provide billable services and support in all areas of RS/6000 SP. His areas of expertise include PSSP, VSD, GPFS, HACMP, TSM, UDB, InfoSpeed and AFS. Theeraphong holds a Bachelor Degree in Computer Engineering from Chulalongkorn University and, as a Monbusho student, a Master Degree in Information Technology from Nagoya Institute of Technology, Japan.

Thanks to the following people for their invaluable contributions to this project:

IBM PPS Lab, Poughkeepsie
Bernard King-Smith

International Technical Support Organization, Poughkeepsie Center
Yoshimichi Kosuge

IBM TJ Watson Research Center
Margaret Momberger

International Technical Support Organization, Austin Center
Diana Groefer, Augie Mena III, Wade Wallace

IBM Poughkeepsie
Bruno Bonetti, Bob Currant, Rich Darvid, Brian Herr, Lyle Gayne, Bob Gensler, Eugene Johnson, Joe Kavaky, Chulho Kim, Gautam Shah, Rajeev Sivaram, Peter Soroka, Clarisse Taaffee-Hedglin, Les Vigotti, James Wang

IBM Hursley
John Hayward, Richard Seymour

IBM Germany
Ewald Vogel

Comments welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in "IBM Redbooks review" on page 515 to the fax number shown on the form.
- Use the online evaluation form found at **ibm.com**/redbooks.
- Send your comments in an Internet note to redbook@us.ibm.com.

Book layout

This redbook is divided into four parts:

Part 1, ”Overview” gives an introduction to SP performance tuning; why is it necessary, when you should tune and how SP tuning differs from tuning a standalone AIX box. It then offers a methodology for tuning - how best to understand the existing environment and how to plan and implement changes to improve performance.

Part 2, ”General Tuning” helps the reader to understand how to improve the general performance of an SP. This does not mean there has to be a known problem in the system. It gives the basic recommended network adapter and system settings to act as a quick reference guide. It then provides a more in depth explanation of individual settings and what affect they have on performance. It also illustrates differences in tuning for different environments, for example, scientific and commercial.

Part 3, “Problem Determination and Tools” provides information on how to detect and solve performance problems that are seen on a system. It starts with a basic checklist to help ascertain where the problem lies, and then goes into greater depth to explain the more common problems seen in an SP environment and how to fix them. This checklist is followed by an explanation of how to best monitor system resources, for example, how to avoid bottlenecks.

Part 4, “Tuning Selected Products” gives tuning information for when the following applications are running on an SP:

- TSM - (ADSM)
- VSD
- GPFS
- Web
- DB2
- Oracle
- Lotus Notes
- Communication Server for AIX

Start map

The START MAP - What to find in this book and where to find it

The intention of this redbook is to give you guidance on either finding or fixing certain performance problems. Perhaps you have no known performance problem, but you want to verify your system settings. To make it easier for you to navigate through this book, you can use the Start Map below. For general information about tuning, when and why, and tuning methodology, you may start with Chapter 1, “Introduction” on page 3; otherwise, take a look at Table 1 on page xxi.

Table 1. Start map

What?	Where?
You want to change your *initial settings*, but you have no idea what to change.	Go to Chapter 3, “Basic performance settings” on page 13. Here you will find some useful initial performance settings.
You want to tune your *network* for better performance.	Go to Chapter 4, “Network tuning” on page 21. Learn to change ARP, MTU and much more.
You want to tune your *adapters* for better performance.	Go to Chapter 5, “Adapter tuning” on page 95. Here you will find SP Switch adapter tuning, Gigabit Ethernet tuning, and much more.
You want to tune components like *Memory, Disk,* and *CPU.*	Go to Chapter 6, “Tuning other components in AIX” on page 115 for more information.
You want to tune TSM, VSD, GPFS, Websphere, DB2, Oracle, and Notes.	Go to Part 4, “Tuning selected products” on page 335 and proceed with the chapters that deal with your specific application.
You simply want *TCPIP + NFS* Overview.	Go to Chapter 4, “Network tuning” on page 21 for more information.
You are looking for *Performance Tools* and how they are used	Go to Chapter 11, “Tools” on page 243 and/or Appendix A, “Performance toolbox for AIX and parallel extensions” on page 469 for more information.
You want to check for performance problems.	Go to Part 3, “Problem determination and tools” on page 187 for more information.
You are looking for the correct command syntax (`no` command).	Go to Section 4.3.2, “AIX tunables” on page 46 for `no` command syntax and much more information.

Part 1. Overview

Chapter 1. Introduction

Welcome to the fully updated “RS/6000 SP System Performance Tuning Guide.” This guide, although still very relevant for lower levels of code, primarily discusses new features included in PSSP version 3.2 and AIX 4.3.3.

In writing this redbook, it is our objective to make it as readable to a novice of performance tuning as it is to an experienced administrator. Considerable thought has gone into the layout of the book. Readers with greater experience in SP tuning have new “quick reference” features, such as startmaps and part guides. This will make referencing parameter settings, tables, and other environment information easier. Full and clear explanation of concepts and procedures are still given, as required.

1.1 Why tune?

Tuning gives us the ability, through the full understanding of our computer environment, to achieve optimal performance.

However, this means we really have to know the goals of the system. What is really critical to achieve those goals? Do we understand what environment we are in? What limitations does the system have that could cause bottlenecks? Only in this way can we begin to prioritize and schedule workloads so that any settings we change have the desired effect.

1.2 When to tune?

Tuning is a continuous process. Computer environments are getting more complex. The SP family, with its growing cluster capabilities, is a good example of this. High speed networking, with concurrent running of jobs, coupled with the ease of scaling that the SP provides means that systems can grow as an organization grows. As more users, applications and data are added, the system has to be tuned and re-tuned so it can absorb the increased load while still delivering the needed performance.

1.3 Why SP tuning differs from normal AIX tuning?

The SP is essentially a centrally managed, network connected collection of servers. It is necessary to carefully consider the differences between the network topology of the nodes and that of standalone AIX boxes.

Systems will have adapters with different transfer rates. Ethernet can be the 10Mb per second or the newer 100Mb per second adapter, so when mixing both, the traffic throughput will be at the slower speed.

The new SP Switch2 is capable of 500 MB per second one way (up from 150 MB per second). If not properly tuned, this could potentially result in one node receiving a volume of data and number of requests exceeding its capacity. The new switch will be discussed in detail in Chapter 5, “Adapter tuning” on page 95.

Other relevant topics are included in:

- Section 4.2.9, “Description of the Nagle Algorithm” on page 42, if the switch is running slow.
- Section 4.2.7.1, “ARP cache tuning” on page 40, for information on caching Address Resolution Protocol (ARP) addresses in SP environments greater than 150 nodes.

Chapter 2. Tuning methodology

Tuning methodology consists of the basic procedures that keep a system well tuned. It is important to see this ongoing process as a *cycle* of events, that requires continual review of the system settings and adjustments for changes in the computing environment.

An important rule to adhere to when tuning or monitoring an SP system is to make sure you have an adequate change control mechanism. You must have clear documentation on what you changed, when you changed it and why you changed it - without this information, any collected monitoring data will not make sense.

2.1 The SP performance tuning cycle

Figure 1 illustrates this performance tuning cycle.

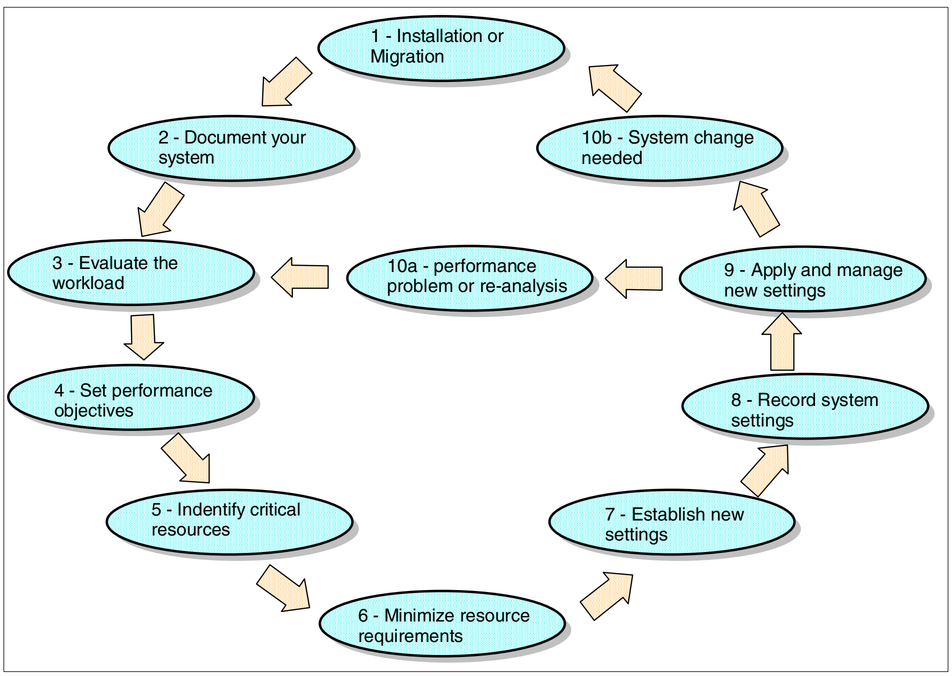

Figure 1. Performance tuning cycle

If you have a well planned and documented existing running system, then you can initially skip steps 1 and 2.

1. Installation or migration.

   Part of an installation or migration is the selection of the appropriate initial tunables for the nodes. IBM provides four alternate tuning files (tuning.cust) that contain initial performance tuning parameters for the following SP environments:

   - Commercial
   - Development
   - Scientific and technical
   - Server

   Homogeneous systems (those systems where the nodes are identical or all have very similar roles) will all need similar or the same tuning. Consolidated systems, where the nodes can have very different roles from each other, present a larger tuning challenge. A full examination is necessary to determine what is running on each node, the traffic through the node, and any interaction between applications. Defining an appropriate set of tunables for a consolidated system requires accounting for everything running on the SP system.

   Grouping together nodes that have identical workloads is a possible solution. These nodes, as a group, usually can use the same set of tunable settings. In the case where there are three distinct node workload types, three sets of tunables need to be determined, and then the appropriate set applied to the tuning.cust file on the appropriate nodes needs to be determined.

   Selecting a tuning file at this stage is not the end of all efforts, but the beginning. These files should be seen as the base setup for getting a running system. You need to go through the rest of the SP performance tuning cycle and adjust the parameters according to your requirements.

2. Document your system.

   When your system is “alive” and installed correctly, you should document your system settings and your network configuration. It is unlikely the system will be running at an optimal performance, but these initial settings will provide a stable environment.

3. Evaluate the workload.

   When planning how to best tune a system, it is important to understand all the work that your system is doing. For example, if your system has file systems that are NFS-mounted and frequently accessed by other systems, handling those accesses could be a significant workload, even though the machine is not officially a server.

With multiuser workloads, the analyst must quantify both typical and peak request rates.

- Components of the workload that have to be identified are:
  - Who will be using the programs running on a system?
  - In which situations will the programs be run?
  - How often do those situations arise, and at what time of day, month or year?
  - Will those situations require changes to existing programs?
  - Which systems will the program run on?
  - How much data will be handled and from where?

Some thought should be given on whether tuning and testing has to be done away from the production environment. Simulating the workload outside a live system is certainly the safest option; an analyst can do more experimental tuning and it does not matter if there is a temporary degradation in performance. However, care has to be taken that this test environment is a true reflection of the production environment. If the workloads are not authentic, any data collected will be inaccurate.

Benchmarking is a popular option in comparing how a particular workload runs on dissimilar systems. These benchmarks can be useful tests if you need to show the relative performance of different systems and configurations, or you feel you do not understand the resource requirements of a workload.

A quality benchmark needs to show relevance to the “real” application and should test what it is intended to test. It needs to be easily portable between platforms and different vendor machines. The results should be meaningful and easily repeatable. The benchmark should be easy to scale for smaller or larger workloads.

The disadvantage of benchmarking is that the workloads used are often very “standardized” in order to allow comparisons between platforms. In many cases, benchmark results will not accurately reflect the performance of a live production system, but can certainly give good indications of potential problems. Also, if the production system is too vast and expensive to duplicate in a test environment, benchmarking can provide a cost effective alternative.

4. Set performance objectives.

   After defining the workload, you can set the objectives. Key to being able to set the objectives is deciding the relative importance of the different work being run on the system. Unless this objective is agreed to in advance, then trade-off decisions between resources will be very difficult. Objectives must be measurable, and this is usually done by expressing the necessary performance as *throughput* and *response time*.

Response time The elapsed time between when a request is submitted and when the response from that request is returned, for example, how long on average does a database query take.

Throughput The number of workload operations that can be accomplished per unit of time, for example, kilobytes of a file transferred per second

5. Identify critical resources.

   Systems have both *real* and *logical* resources. Real resources are the physical components, such as disk space, network adapters, memory and CPU. Logical resources are abstractions that enable the real resources to be shared. An example of a CPU logical resource is a processor time slice. Memory logical resources include page frames, buffers, locks and semaphores. Disk space is divided into volume groups, logical volumes and file systems, while networking has sessions packets and channels.

   If several applications are sharing the same node they will be in contention for, for example, CPU. The user process priorities can be manipulated using the `nice` or `renice` commands and the `schedtune` command can change the time slice, although this is for all processes. Applications that may not or should not have the whole time slice can give up the processor time slot with the `yield` system call. The new *Workload Manager* that is in AIX 4.3.3 can also help with prioritizing workload.

6. Minimize the workload's critical resource requirements.

   To minimize the critical resource requirements of a workload, conscious decisions have to be taken as to where applications will run and whether, for example, files should be kept locally or on a server. Basic administrative tasks, such as clearing log files or backing up file systems, could cause network traffic that could affect system performance. Scheduling such tasks via the crontab is a good way to make sure these important tasks do not get forgotten and also run at quieter times.

   If you already understand what the CPU or memory peak times on a system are at a system-management level, low priority workloads that are contending for critical resource can be moved to other systems, run at other times or controlled with the Workload Manager.

   The parallelism of the SP environment can help the resource problem considerably, although the nodes' resources, such as memory and CPU, are separate. Internode bandwidth means that jobs can run on different nodes, concurrently passing needed data between nodes. Software that runs in this kind of environment, such as POE and LoadLeveler, will use the Ethernet as well as the Switch, and a peak performance may not be obtainable on both at the same time.

7. Establish new settings to reflect resource priorities.

   Now you can plan what actual settings to change. This could mean changing the *AIX network tunables*. The `no` command is used to display and change these values (these values will be discussed in detail in Chapter 4, “Network tuning” on page 21). Adapter tuning is discussed in detail in Chapter 5, “Adapter tuning” on page 95 and includes, for example, information on sizing adapter and switch adapter pools.

8. Record and keep track of system settings.

   Record all changes that have been made. You must know what was changed, when it was changed, when data was collected and what this data was good for. Along with the change records, keep a log of the performance impacts on the system and the nodes during the changes.

9. Apply and manage new settings

   It is important to know where to set these new tunable values. If they are not set in the correct places, you may not use the changed settings. In the worst case, the node will not reboot at all.

   For all dynamic tunables (those that take effect immediately), the settings for each node should be set in its local /tftpboot/tuning.cust file. It is guaranteed that this file will be executed on every reboot. Tunables changed using the `no`, `nfso`, or `vmtune` commands can be included in this file.

   If the system has nodes that require different tuning settings, it is recommended that a copy of each of the settings be saved on the control workstation; then when nodes of a specific tuning setting are installed, that version of tuning.cust can be moved into /tftpboot on the control workstation. You can amalgamate these tuning.cust files into one by reading each node’s node number from the SDR, then, based on the node number, set the appropriate values for that node. This avoids maintaining several different tuning.cust files and making sure the correct one is loaded on each node.

10. Monitor performance indices.

    Continue to check the system and monitor the performance indices (such as response time of applications, throughput, possible errors, and so on) to prove that the changes led to an overall improvement.

Resource monitoring is dealt with in Chapter 10, “Resource monitoring” on page 203, which gives an insight into how best to monitor memory, CPU, and I/O resources. A further chapter Chapter 11, “Tools” on page 243 gives details of general tools for monitoring that are available, for example, `topas`.

Eventually, one of the following two situations may occur.

a. Performance problem or re-analysis necessary.

The system shows performance problems, or the analyst believes better performance can be achieve by readjusting some parameters. Record and analyze the data that shows any performance degradation. Perform AIX performance analysis, SP Switch analysis, SP log analysis, and so forth, and then go back to Step 3.

b. System changes

Some hardware or software is going to be added, replaced by a newer version or removed from the system. Note the changes, identify node role changes (if any), then go back to Step 1.

Part 2. General tuning

Chapter 3. Basic performance settings

In this chapter, we outline basic performance tuning settings that you always should apply to all nodes in a RS/6000 SP (and most other large RS/6000 systems), as well as the Control Workstation (CWS). These settings are the starting point from which performance monitoring and tuning begins. If you perceive that the RS/6000 SP system is not performing as you desire or expected, make sure that these values have been set on all systems in your SP complex before investigating further. These values are based on SP experts experiences in dealing with both real life customer workloads and in advanced benchmarking scenarios.

View these settings as the basics that should always be applied on all customer production systems. Remember that the default values on many of the values for the logical resources are inherited from AIX v3 and are backwards compatible with hardware that is still supported but designed and manufactured over ten years ago. The default values are also intended for workstation workloads and not for large multiuser, multiprocessor systems.

We give you tables and command examples of how to set these basic values. If you manage more than one RS/6000 SP node or networked RS/6000 system, you should use a shell script to set the values, so that none are overlooked and missed, or you should use a checklist where you set values, and after each reboot check the setting in the production environment.

On the RS/6000 SP you can change both the devices or just put all values into `/tftpboot/tuning.cust` on each node. If you maintain a good `/tftpboot/script.cust`, you could divide the settings for the nodes into one part that is done by the installation/customization and one part that is done at boot (IPL) time (`first.boot`). If you do not use a well documented and structured `/tftpboot/script.cust`, set all values in `/tftpboot/tuning.cust`. This latter method is also good for when you apply these changes to a running production system.

Important

1. Document current values before changing them.
2. Do not type any errors (use shell scripts if possible).
3. Pretype the command to reset the values using another terminal connection just in case something goes wrong.

3.1 Basic network settings

Before applying the basic performance tuning settings, document the current value of each, as in the following example:

```
# no -a | tee /tftpboot/no.$(date +”%Y%m%d”).out
```

When this is done, create a script or update `/tftpboot/tuning.cust` so that these values are set at every boot (IPL). Use the following as an example:

```
# no -o sb_max=1310720
```

To set all `no` tunables back to their default value, use the following command:

```
# no -a | awk '{print $1}'|xargs -t -i no -d {}
```

The values in Table 2 are to be used as a starting point. For information on these parameters, refer to Section 4.3.2, “AIX tunables” on page 46.

Table 2. Basic network settings with the no command

no parameter	Value	Comment
thewall	Use the *default*	If network errors occur, set manually to 1048500
sb_max	1310720	sb_max limits what each socket can get out of thewall
tcp_sendspace	262144	sb_max / 5
tcp_recvspace	262144	sb_max / 5
udp_sendspace	65536	sb_max / 20
udp_recvspace	655360	sb_max / 2
tcp_mssdflt	1448	1448 for Ethernet and 1440 for Token-Ring
rfc1323	1	
ipqmaxlen	512	
tcp_pmtu_discover	0	
udp_pmtu_discover	0	

To document all the network interfaces’ important device settings, you can manually check all interface device drivers with the `lsattr` command, or run the following script:

```
#!/bin/ksh

stripit()
{
  awk '
  # example: 1...999
  /\.\.\./{print $1;next}
  # example: "attribute=1-999"
  / "/{a=substr($0,index($0,"\"")+1);
    b=substr(a,index(a,"=")+1);
    print substr(b,0,index(b,"\"")-1);next}'|read value

  if [[ ! -z "$value" ]];then
    print "$value"
  else
    print "N/A"
  fi
}

printf "%-6.6s %-16.16s %-16.16s %-s\n" "Device" "Attribute" "Value" "Range"
for device in $(lsdev -F"name:subclass" -C -c if|grep -v LO|cut -f1 -d:);do
  for attribute in netaddr mtu remmtu rfc1323 tcp_nodelay tcp_sendspace tcp_recvspace
tcp_mssdflt;do
   lsattr -F"value" -E -l $device -a $attribute|awk '{print ($1)?$1:"none"}'|read value
   lsattr -R -l $device -a $attribute 2>&1 |stripit|read range
   printf "%-6.6s %-16.16s %-16.16s %-s\n" "$device" "$attribute" "$value" "$range"
 done
done
```

3.2 Basic adapter settings

Network adapters should be set to utilize the maximum transfer capability of the current network. Maximum values for device driver buffers and caches should be set.

To find out the maximum possible setting for a device, use the `lsattr` command as will be shown in the following examples. First find out the attribute names of the device driver buffers/queues that the adapter uses:

```
# lsattr -El ent0
busio           0x7ff800        Bus I/O address                  False
busintr         7               Bus interrupt level              False
intr_priority   3               Interrupt priority               False
tx_que_size     2048            TRANSMIT queue size              True
rx_que_size     256             RECEIVE queue size               True
rxbuf_pool_size 2048            RECEIVE buffer pool size         True
media_speed     100_Full_Duplex Media Speed                      True
use_alt_addr    no              Enable ALTERNATE ETHERNET address True
alt_addr        0x000000000000  ALTERNATE ETHERNET address       True
ip_gap          96              Inter-Packet Gap                 True
```

Then find out the range for each queue as follows:

```
# lsattr -Rl ent0 -a tx_que_size
16...2048 (+1)

# lsattr -Rl ent0 -a rx_que_size
16
32
64
128
256

# lsattr -Rl ent0 -a rxbuf_pool_size
16...2048 (+1)
```

For the `tx_que_size` and `rxbuf_pool_size`, the maximum is 2048; for `rx_que_size`, the maximum is 256. To change the values so that they will be used the next time the device driver is loaded, use the `chdev` command, as shown here:

```
# chdev -l ent0 -a tx_que_size=2048 -a rxbuf_pool_size=2048 -a rx_que_size=256 -P
ent0 changed
```

The following script extracts the maximum values allowed for Ethernet and Token-Ring adapters (from the ODM) to use for the transmit and the receive queues (respectively):

```
#!/bin/ksh

execute=$1

stripit2()
{
 awk '
  # example: 1...999
  /\.\.\./{x=substr($1,index($1,".")+3)}
  # example: "attribute=1-999"
  /\=/{a=substr($0,index($0,"=")+1);
    b=substr(a,index(a,"-")+1);
    x=substr(b,0,index(b,"\"")-1)}
  # example: 16\n32\n64\n256
  NR>1{x=$1}
  END{print x}'
}

types="ent|tok"

printf "%-6.6s %-16.16s %-8.8s %-8.8s %-s\n" "Device" "Attribute" "Value" "Maxvalue"
"Command"

for device in $(lsdev -Cc adapter -Fname|egrep $types);do
  for attribute in $(lsattr -E -Fattribute -l $device|grep "_.*_size");do
    lsattr -E -F"value" -l $device -a $attribute 2>&1 |read value
    lsattr -R -l $device -a $attribute 2>&1 |stripit2|read maxvalue
    cmd="chdev -l $device -a $attribute=$maxvalue"
    printf "%-6.6s %-16.16s %-8.8s %-8.8s %s\n" "$device" "$attribute" "$value" "$maxval
"$cmd"
    if [[ ! -z "$execute" ]];then
      $cmd -P  # Sets the change in the ODM only
    fi
  done
done
```

To set the maximum values with the above script, just start it with a parameter. For example (if the script above was named `change_adapter_buffers`):

```
# change_adapter_buffers 1
```

If used without a parameter, it would show all the maximum values for all attributes for all Ethernet and Token-Ring adapters in the system.

3.3 Basic AIX system settings

Do not make any memory threshold changes until you have had experience with the response times of the system for the actual workload.

Use the default CPU scheduling parameters, unless you have extensive monitoring and tuning experience with the same workload on a similar configuration. Leave most of the AIX system settings for memory management unchanged at installation time.

The mechanisms for defining and expanding logical volumes attempt to make the best possible default choices. However, satisfactory disk I/O performance is much more likely if the installer of the system tailors the size and placement of the logical volumes to the expected data storage and workload requirements. But this is very dependent on the available hardware configuration and should have been considered at the design stage of the system layout. When creating multiple large logical volumes that span multiple disks, start by using the *edge* inter-disk allocation policy, and let AIX create the logical volumes consecutively starting from the edge[1].

You should consider increasing the `maxuproc` setting to reflect the maximum number of processes that will be allowed per user; otherwise, some applications might not run properly (the default of 40 is usually to low). You should also change `autorestart` to true so that the system will restart if it crashes.

```
# chdev -l sys0 -a maxuproc=1024 -a autorestart=true
```

Also, run the `schedtune` and `vmtune` commands to document the initial settings for the *Scheduler* and the *Virtual Memory Manager (VMM)*. These commands can be found in the *bos.adt* sample fileset.

```
# /usr/samples/kernel/schedtune
# /usr/samples/kernel/vmtune
```

Note that you will very probably need to change the VMM settings for `minperm`/`maxperm` and `minfree`/`maxfree` with `vmtune`. For more information, refer to Section 6.1.2.6, “minperm and maxperm” on page 123 and Section 6.1.2.4, “minfree and maxfree” on page 119.

For nodes that will have a lot of networked users logged in or applications running that use pseudo terminals, you will need to increase the maximum number of PTYs allowed on the system using the following command:

[1] If allocated from the center, there is a risk that the logical volumes, after the first one, will get split around the previous logical volume(s) that occupy most of the center already.

```
# chdev -l pty0 -a BSDnum=128
```

The default for `BSDnum` is 16 (and 256 for `ATTnum`).

Chapter 4. Network tuning

The tuning objectives for the RS/6000 SP are: improve performance, response time, and resource utilization. These objectives look similar to those for a stand-alone RS/6000 system. Nevertheless, the approach for tuning an SP system is, in some situations, different from how you would tune an AIX workstation.

This chapter discusses the most important parameters involved in a general network tuning and how these same parameters are tuned for in a RS/6000 SP system. We will also describe specific tuning strategies for SP systems in different environments.

4.1 Initial considerations

The basic architecture of the SP is a set of nodes connected by a communication layer. Therefore, the most important aspect of tuning concerns the communication network. Once the RS/6000 SP communication layer is properly tuned, use standard AIX tuning within the nodes.

4.1.1 General tuning recommendations

The very first step always involves monitoring the system. Keeping a detailed log (statistics and also parameters) of your system before and after any configuration changes could save hours of distress later. Any changes to your SP environment, whether you are adding applications or changing your subsystems, requires a full review of all your system parameters.

In tuning an AIX workstation or server, the most common approach is to tune the machine to handle the amount of traffic or services requested of it. In the case of a file server, the server is tuned to the maximum amount of traffic it can receive. In general, the bottleneck in a high-end server is the capacity of the network through which services are requested.

The situation with an SP system could be the opposite in some situations. The SP Switch is faster than any other network available. With the non-blocking nature of a switch, the number of requests and volume of data that may be requested of a node can far exceed the node's capacity. To properly handle this situation, the SP system must manage the volume of services requested of a server. In other words, you should reduce the number of requests at the client rather than increase the capacity of the server. It is very easy on large SP system configurations to require more services than the most powerful node can currently deliver.

4.1.2 Consolidated system challenges

Consolidated systems present a larger tuning challenge than homogeneous systems. *Consolidated systems* are those in which different nodes assume different roles. In such a case, make a full examination of what is running on each node, the traffic through the node, and any interaction between applications. Defining an appropriate set of tunables for a consolidated system requires accounting for everything running on each node in the SP system.

The typical method of picking apart the consolidated tuning problem is to group nodes with identical workloads. These nodes, as a group, usually can use the same set of tunables. Where there are three distinct node workload types, three sets of tunables need to be determined and the appropriate set applied to the tuning.cust file on the appropriate nodes.

4.1.3 System topology considerations

The most important consideration in configuring the SP Ethernet is the number of subnets configured. The routing through the Ethernet can be complicated because of the limitation on the number of simultaneous network installs per subnet. More information regarding the topology of the Ethernet can be found in *RS/6000 SP: Planning Volume 1, Hardware and Physical Environment,* GA22-7280.

Systems that use Ethernet switches need to address the flat Ethernet topology rather than a hierarchical network tree of the traditional SP Ethernet.

When configuring software that uses the SP Ethernet, consider the software location in the SP Ethernet topology. Software such as LoadLeveler and POE use the SP Ethernet to communicate to other nodes. Installing such software on a far point in the network in a large SP configuration can cause bottlenecks on the network subnets and the adapters connected to the CWS. On systems where the CWS is the default route, the Ethernet adapter that maps to the default address can become a bottleneck as traffic is all routed through this one adapter.

Installing such software on the CWS causes the lowest possible traffic. However, the CWS has to be powerful enough to act as the CWS and, in addition, support the additional software and network traffic.

It is not recommended that the SP Ethernet be local to other parts of the outside computing network topology. Keeping only SP traffic on the SP Ethernet prevents outside network traffic from causing performance problems on the SP itself. If you have to connect the SP Ethernet to your external network, make sure that the outside traffic does not overload the SP Ethernet. You can overload it if you route high-speed network adapter traffic (for example, FDDI or ATM) through the SP Ethernet. Route gateway traffic over the SP switch from gateways to FDDI, ATM, and other high-speed networks. Configure routers or gateways to distribute the network traffic so that one network or subnet is not a bottleneck.

Several gateways or the SP router node should be configured if a large volume of traffic is expected. All traffic on these networks can be monitored using the standard network monitoring tools. Details about the monitoring tools and their usage can be found in *Understanding IBM RS/6000 Performance and Sizing, SG24-4810* or in Section 10.5, "Monitoring Network I/O" on page 231.

4.2 TCP/IP network overview for performance

In the following pages, there is a description of the most important TCP/IP concepts related to performance. For more information about TCP/IP, refer to *Understanding IBM RS/6000 Performance and Sizing, SG24-4810* and *AIX V4.3 System Management Guide: Communications and Networks*, SC23-4127.

TCP/IP carefully defines how information moves from sender to receiver:

1. First, application programs send messages or streams of data to one of the Internet Transport Layer Protocols, either the User Datagram Protocol (UDP) or the Transmission Control Protocol (TCP).
2. These protocols receive the data from the application, divide it into smaller pieces called *packets*, add a packet header (including the destination address), and then pass the packets along to the next protocol layer (the Internet Network layer). The TCP/IP transport-level protocols allow application programs to communicate with other application programs. UDP and TCP are the basic transport-level protocols for making connections between Internet hosts. Both TCP and UDP use protocol ports on the host to identify the specific destination of the message.
3. The Internet Network layer encloses the packet in an Internet Protocol (IP) datagram, puts in the datagram header and trailer, decides where to send the datagram (either directly to a destination or else to a gateway), and passes the datagram on to the Network Interface layer.

4. The Network Interface layer accepts IP datagrams and transmits them as frames over a specific network hardware, such as Ethernet or Token-Ring networks.
5. Frames received by a host go through the protocol layers in reverse. Each layer strips off the corresponding header information, until the data is back at the application layer (see Figure 2 on page 25). Frames are received by the Network Interface layer (in this case, an Ethernet adapter). The Network Interface layer strips off the Ethernet header, and sends the datagram up to the Network layer. In the Network layer, the Internet Protocol strips off the IP header and sends the packet up to the Transport layer. In the Transport layer, the Transmission Control Protocol (in this case) strips off the TCP header and sends the data up to the Application layer.

Higher-level protocols and applications use UDP to make datagram connections and TCP to make stream connections. The operating system sockets interface implements these protocols.

4.2.1 Send and receive flows in the TCP/IP protocol stack

In this section, we describe, in more detail, the send and receive flows that go through the TCP/IP stack. Figure 2 on page 25 is an illustration of all of these flows.

In the following paragraphs you will find some variables or parameters that you can adjust in AIX environments for performance reasons. For a more exhaustive description of the AIX network tunables go to Section 4.3, “AIX network tunables” on page 45.

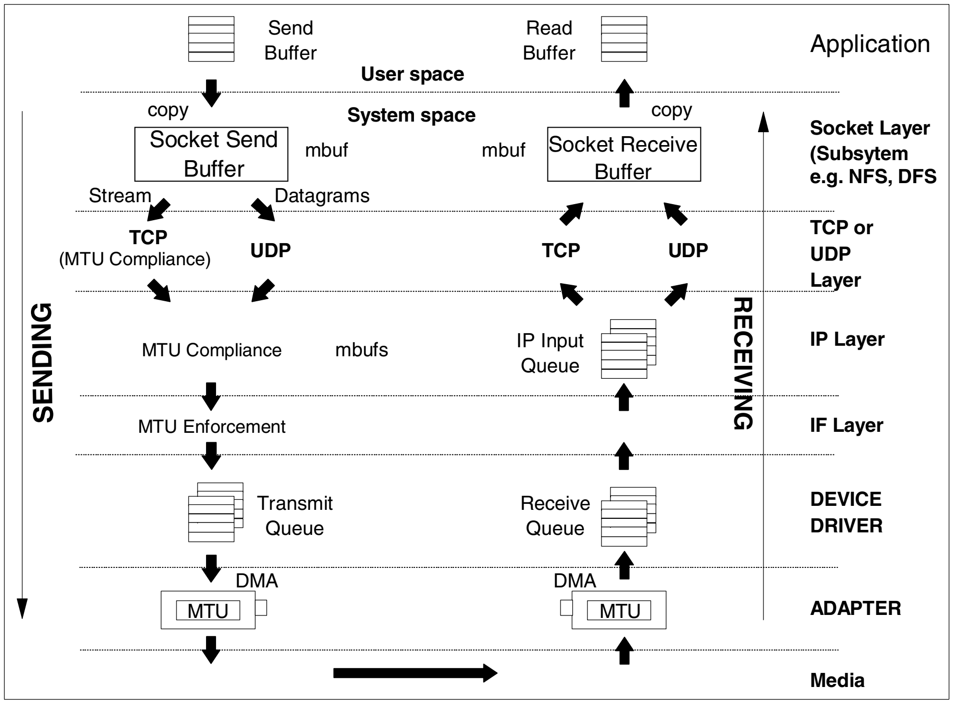

Figure 2. TCP/UDP/IP data flow

4.2.1.1 UDP send flow

If udp_sendspace (size of the UDP socket send buffer) is large enough to hold the datagram, the application's data is copied into mbufs in kernel memory. If the datagram is larger than udp_sendspace, an error is returned to the application. The operating system chooses optimum size buffers from a power of 2 size buffer. For example, a write of 8704 bytes is copied into two clusters, a 8192-byte and a 512-byte cluster. UDP adds the UDP header (in the same mbuf, if possible), checksums the data, and calls the IP ip_output() routine.

4.2.1.2 UDP receive flow

UDP verifies the checksum and queues the data onto the proper socket. If the udp_recvspace (size of the UDP socket receive buffer) limit is exceeded, the packet is discarded. A count of these discards is reported by the `netstat -s` command under udp label as socket buffer overflows.

```
# netstat -s
udp:
        2844603 datagrams received
        0 incomplete headers
        0 bad data length fields
        0 bad checksums
        1027 dropped due to no socket
        140406 broadcast/multicast datagrams dropped due to no socket
        0 socket buffer overflows
        2703170 delivered
        2514585 datagrams output
```

If the application is waiting for a receive or read on the socket, it is put on the run queue. This causes the receive to copy the datagram into the user's address space and release the mbufs, and the receive is complete. Normally, the receiver responds to the sender to acknowledge the receipt and also return a response message.

4.2.1.3 TCP send flow

When the TCP layer receives a write request from the socket layer, it allocates a new mbuf for its header information and copies the data in the socket-send buffer either into the TCP-header mbuf (if there is room) or into a newly allocated mbuf chain. If the data being copied is in clusters, the data is not actually copied into new clusters. Instead, a pointer field in the new mbuf header (this header is part of the mbuf structure and is unrelated to the TCP header) is set to point to the clusters containing the data, thereby avoiding the overhead of one or more 4 KB copies.

TCP then checksums the data (unless it is off-loaded by certain PCI adapters), updates its various state variables, which are used for flow control and other services, and finally calls the IP layer with the header mbuf now linked to the new mbuf chain.

4.2.1.4 TCP receive flow

When the TCP input routine receives input data from IP, the following occurs:

1. It checksums the TCP header and data for corruption detection (unless it is off-loaded by certain PCI adapters).
2. Determines which connection this data is for.
3. Removes its header information.
4. Links the mbuf chain onto the socket-receive buffer associated with this connection.
5. Uses a socket service to wake up the application (if it is sleeping).

4.2.1.5 IP send flow

When the IP output routine receives a packet from UDP or TCP, it identifies the interface to which the mbuf chain should be sent, updates and checksums the IP part of the header, and passes the packet to the interface (IF) layer. IP determines the proper device driver and adapter to use based on the network number. The driver interface table defines the maximum MTU for this network. If the datagram is less than the MTU size, IP adds the IP header in the existing mbuf, checksums the IP header, and calls the driver to send the frame. If the driver send queue is full, an EAGAIN error is returned to IP, which returns it to UDP, which returns it to the sending application. The sender should delay and try again.

If the datagram is larger than the MTU size (which only occurs in UDP), IP fragments the datagram into MTU-size fragments, appends an IP header (in an mbuf) to each, and calls the driver once for each fragment frame. If the driver's send queue is full, an EAGAIN error is returned to IP. IP discards all remaining unsent fragments associated with this datagram and returns EAGAIN to UDP. UDP returns EAGAIN to the sending application. Since IP and UDP do not queue messages, it is up to the application to delay and try the send again.

4.2.1.6 IP receive flow

Interfaces do not perform queuing and directly call the IP input queue routine to process the packet; the loopback interface will still perform queuing. In the case of queuing, the demux layer places incoming packets on this queue. If the queue is full, packets are dropped and never reach the application. If packets are dropped at the IP layer, the statistic called ipintrq overflows in the output of the `netstat -s` command is incremented. If this statistic increases in value, then use the `no` command to tune the ipqmaxlen tunable.

```
# netstat -s
ip:
          0 path MTU discovery memory allocation failures
        0 ipintrq overflows
        0 with illegal source
        0 packets dropped by threads
        0 packets dropped by threads
```

The demux layer calls IP on the interrupt thread. IP checks the IP header checksum to make sure the header was not corrupted and determines if the packet is for this system. If so, and the frame is not a fragment, IP passes the mbuf chain to the TCP or UDP input routine.

If the received frame is a fragment of a larger datagram (which only occurs in UDP), IP retains the frame. When the other fragments arrive, they are merged into a logical datagram and given to UDP when the datagram is complete. IP holds the fragments of an incomplete datagram until the ipfragttl time (as specified by the `no` command) expires. The default ipfragttl time is 30 seconds (an ipfragttl value of 60 half-seconds). If any fragments are lost due to problems such as network errors, lack of mbufs, or transmit queue overruns, IP never receives them. When ipfragttl expires, IP discards the fragments it did receive. This is reported as a result from the `netstat -s` command under ip label as fragments dropped after timeout.

```
# netstat -s
ip:

        35 fragments dropped after timeout
```

4.2.1.7 Demux send flow

When the demux layer receives a packet from IP, it attaches the link-layer header information to the beginning of the packet, checks the format of the mbufs to make sure they conform to the device driver's input specifications, and then calls the device driver write routine. The address resolution protocol (ARP) is also handled in this layer. ARP translates a 32-bit Internet address into a 48-bit hardware address. ARP is discussed in Section 4.2.7, “Address resolution” on page 40.

4.2.1.8 Demux receive flow

When the demux layer receives a packet from the device driver, it calls IP on the interrupt thread to perform IP input processing. If the “dog threads” are enabled, the incoming packet will be queued to the thread and the thread will handle calling IP, TCP, and the socket code. Refer to Section 4.2.4, “Enabling thread usage on LAN adapters (dog threads)” on page 32 for more information.

4.2.2 MTU

The Maximum Transmission Unit (MTU) specifies the maximum size of a packet (including all the protocol headers) that can be transmitted on a network. All nodes and/or systems on the same physical network must have the same MTU. The MTU can be displayed using the `netstat -i` command. Table 3 on page 29 gives an overview of common network adapters and their related MTU sizes.

For two hosts communicating across a path of multiple networks, a transmitted packet will become fragmented if its size is greater than the smallest MTU of any network in the path. Since packet fragmentation can result in reduced network performance, it is desirable to avoid fragmentation by transmitting packets with a size that is no greater than the smallest MTU in the network path. This size is called the path MTU.

Table 3. Maximum transmission units

Network Type	Default MTU	Maximum MTU	Optimal
Ethernet	1500	1500	1500
Gigabit Ethernet	1500	9000	9000
Token Ring	1492	17284	4096 8500 for NFS traffic
Escon	1500	4096	4096
FDDI	4352	4352	4352
ATM	9180	65530	9180
HiPS	65520	65520	65520
SP Switch	65520	65520	65520
SP Switch2	65520	65520	65520
HiPPI	65536	65536	65536

The MTU value can be changed per adapter using the `ifconfig` command or via SMIT. Because all systems on the same physical network should have the same MTU, any changes made should be made simultaneously. The change is effective across system boots.

4.2.3 Communication subsystem memory (mbuf) management

This section describes the mechanism used by AIX to manage the memory dedicated to the communication subsystem.

4.2.3.1 Mbufs

To avoid fragmentation of kernel memory and the overhead of numerous calls to the xmalloc() subroutine, the various layers of the communication subsystem share common buffer pools. The mbuf management facility controls different buffer sizes that can range from 32 bytes up to 16384 bytes.

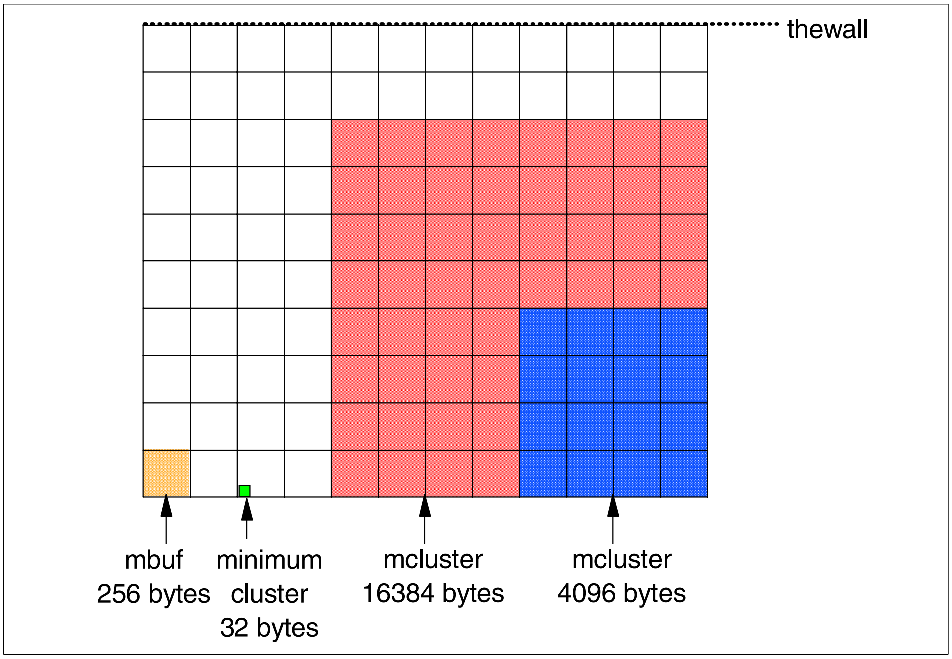

Figure 3. The network memory pool

The pools are created from system memory by making an allocation request to the Virtual Memory Manager (VMM). The pools consist of pinned pieces of kernel virtual memory in which they always reside in physical memory and are never paged out. The result is that the real memory available for paging in application programs and data has been decreased by the amount that the mbuf pools have been increased.

The network memory pool is split evenly among each processor, as show in Figure 3. Each sub-pool is then split up into buckets of 32-16384 bytes. Each bucket can borrow memory from other buckets on the same processor but a processor cannot borrow memory from another processor's network memory pool. When a network service needs to transport data, it can call a kernel service such as *m_get()* to obtain a memory buffer. If the buffer is already available and pinned, it can get it immediately. If the upper limit has not been reached, and the buffer is not pinned, then a buffer is allocated and pinned. Once pinned, the memory stays pinned, but can be freed back to the network pool. If the number of free buffers reaches a high-water mark, then a certain number is unpinned and given back to the system for general use.

An upper limit is placed on how much of real memory (RAM) can be used for network memory. You can tune this parameter by setting maxmbuf or thewall. For more details about these parameters go to Section 4.3, “AIX network tunables” on page 45.

4.2.3.2 Sockets

Sockets provide the application program interface (API) to the communication subsystem. Several types of sockets provide various levels of service by using different communication protocols. Sockets of type SOCK_DGRAM use the UDP protocol. Sockets of type SOCK_STREAM use the TCP protocol. See Figure 4 for an overview.

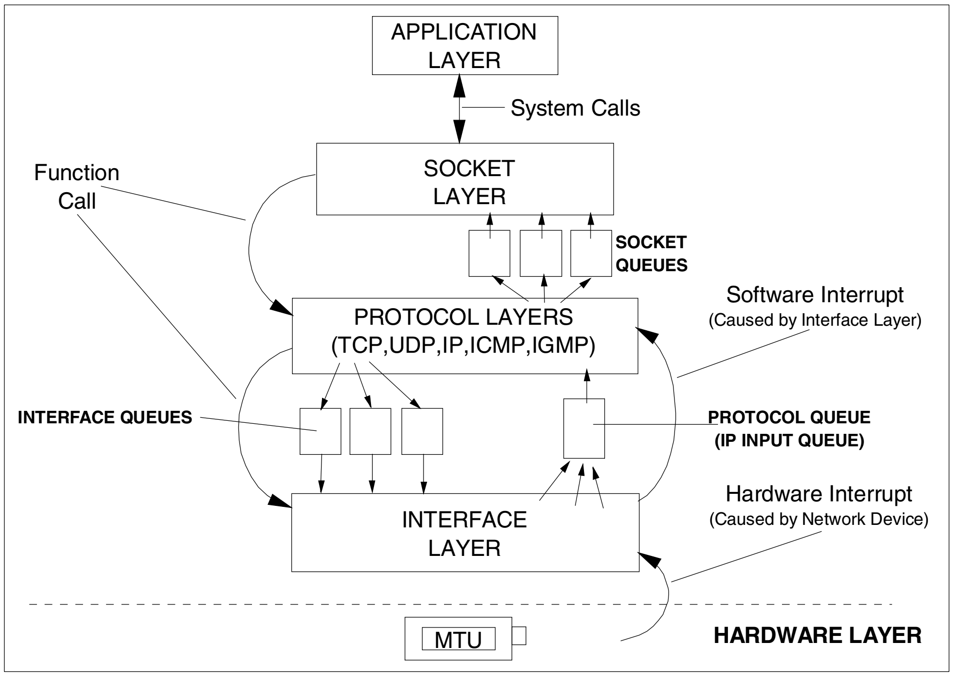

Figure 4. Socket layer

The semantics of opening, reading, and writing to sockets are similar to those for manipulating files.

The sizes of the buffers in system virtual memory (that is, the total number of bytes from the mbuf pools) that are used by the input and output sides of each socket are limited by system-wide default values (which can be overridden for a given socket by a call to the setsockopt() subroutine).

4.2.4 Enabling thread usage on LAN adapters (dog threads)

Drivers, by default, call IP directly, which calls up the protocol stack to the socket level while running on the interrupt level. This minimizes instruction path length, but increases the interrupt hold time. On an SMP system, a single CPU can become the bottleneck for receiving packets from a fast adapter. By enabling the dog threads, the driver queues the incoming packet to the thread and the thread handles calling IP, TCP, and the socket code. The thread can run on other CPUs which may be idle. Enabling the dog threads can increase capacity of the system in some cases.

> **Note**
>
> This feature is not supported on uniprocessors, because it would only add path length and slow down performance.

This is a feature for the input side (receive) of LAN adapters. It can be configured at the interface level with the `ifconfig` command in two ways:

```
# ifconfig interface thread
# ifconfig interface hostname up thread
```

To disable the feature, use the following command:

```
# ifconfig interface -thread
```

The interface parameter used in the `ifconfig` command must be replaced with the interface type you want to configure. For example, use *en0* for Ethernet.

When considering using *dog threads*, the guidelines are as follows:

- More CPUs than adapters need to be installed. Typically, at least two times more CPUs than adapters are recommended.
- Systems with faster CPUs benefit less. Machines with slower CPU speed may help the most.
- This feature is most likely to improve performance when there is high input packet rate. It will improve performance more on MTU 1500 compared to MTU 9000 (jumbo frames) on Gigabit, as the packet rate will be higher on small MTU networks.
- The dog threads run best when they find more work on their queue and do not have to go back to sleep (waiting for input). This saves the overhead of the driver waking up the thread and the system dispatching the thread.

The dog threads can also reduce the amount of time a specific CPU spends with interrupts masked. This can release a CPU to resume normal user-level work sooner.

The dog threads can also reduce performance by about 10 percent if the packet rate is not fast enough to allow the thread to keep running. The 10 percent is an average amount of increased CPU overhead needed to schedule and dispatch the threads.

4.2.5 Tuning TCP maximum segment size

The Maximum Segment Size (MSS) is the largest segment or *chunk* of data that TCP will send to a destination. The TCP protocol includes a mechanism for both ends of a connection to negotiate the MSS to be used over the connection. Each end uses the *OPTIONS* field in the TCP header to advertise a proposed MSS. The MSS that is chosen is the smaller of the values provided by the two ends.

The purpose of this negotiation is to avoid the delays and throughput reductions caused by fragmentation of the packets when they pass through routers or gateways and reassemble at the destination host.

The value of MSS advertised by the TCP software during connection setup depends on whether the other end is a local system on the same physical network (that is, the systems have the same network number) or whether it is on a different (remote) network.

4.2.5.1 Local network

If the other end of the connection is local, the MSS advertised by TCP is based on the MTU (maximum transmission unit) of the local network interface, as follows:

```
TCP MSS = MTU - TCP header size - IP header size.
```

Because this is the largest possible MSS that can be accommodated without IP fragmentation, this value is inherently optimal, so no MSS tuning is required for local networks.

4.2.5.2 Remote network

When the other end of the connection is on a remote network, this operating system's TCP defaults to advertising an MSS of 512 bytes. This conservative value is based on a requirement that all IP routers support an MTU of at least 576 bytes. The optimal MSS for remote networks is based on the smallest MTU of the intervening networks in the route between source and destination. In general, this is a dynamic quantity and could only be ascertained by some form of path MTU discovery. The TCP protocol, by default, does not provide a mechanism for doing path MTU discovery, which is why a conservative MSS value is the default.

However, it is possible to enable the TCP MTU discovery by using the following command (MTU path discovery default is on in AIX 4.3.3):

```
# no -o tcp_pmtu_discover=1
```

A normal side effect of this setting is to see the routing table increase (one more entry per each active TCP connection). The `no` option route_expire should be set to a non-zero value, in order to have any unused cached route entry removed from the table, after route_expire time of inactivity.

While the conservative default is appropriate in the general Internet, it can be unnecessarily restrictive for private intranets within an administrative domain. In such an environment, MTU sizes of the component physical networks are known, and the minimum MTU and optimal MSS can be determined by the administrator. The operating system provides several ways in which TCP can be persuaded to use this optimal MSS. Both source and destination hosts must support these features. In a heterogeneous, multi-vendor environment, the availability of the feature on both systems can determine the choice of solution.

To override the MSS default using the tcp_mssdflt option, with the `no` command or on the adapter, specify a value that is the minimum MTU value less 40 to allow for the normal length of the TCP and IP headers. The size is the same as the MTU for communication across a local network with one exception: the size is only for the size of the data in a packet. Reduce the tcp_mssdflt for the size of any headers so that you send full packets instead of a full packet and a fragment.

Calculate MSS

MTU of interface - TCP header size - IP header size - rfc1323 header size

(MTU - 20 - 20 - 12, or MTU - 52)

Limiting data to (MTU - 52) bytes ensures that, where possible, only full packets will be sent.

In an environment with a larger-than-default MTU, this method is advantageous because the MSS does not need to be set on a per-network basis. The disadvantages are as follows:

- Increasing the default can lead to IP router fragmentation if the destination is on a network that is truly remote and the MTUs of the intervening networks are not known.
- The tcp_mssdflt parameter must be set to the same value on the destination host.

4.2.5.3 Subnetting

Several physical networks can be made to share the same network number by subnetting. The `no` option subnetsarelocal specifies, on a system wide basis, whether subnets are to be considered local or remote networks. With the command `no -o subnetsarelocal=1` (the default), Host A on subnet 1 considers Host B on subnet 2 to be on the same physical network.

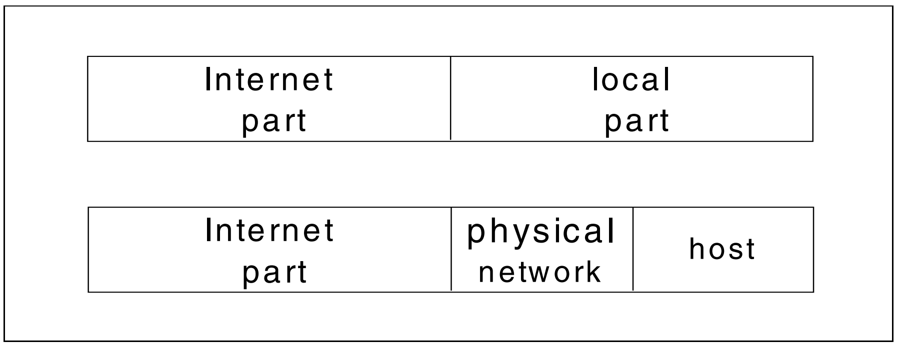

Figure 5. Subnet addressing

The consequence is that when Host A and Host B establish a connection, they negotiate the MSS assuming they are on the same network. Each host advertises an MSS based on the MTU of its network interface, usually leading to an optimal MSS being chosen.

Figure 6 on page 36 gives us an overview of this process

The advantages of this approach are as follows:

- It does not require any static bindings; MSS is automatically negotiated.
- It does not disable or override the TCP MSS negotiation, so that small differences in the MTU between adjacent subnets can be handled appropriately.

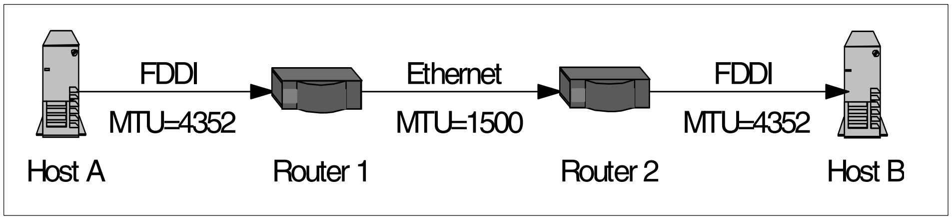

Figure 6. Inter-subnet fragmentation

The disadvantages of this approach are as follows:

- Potential IP router fragmentation when two high-MTU networks are linked through a lower-MTU network.
- Hosts A and B would establish a connection based on a common MTU of 4352. A packet going from A to B would be fragmented by Router 1 and defragmented by Router 2. The reverse would occur going from B to A.
- Source and destination must both consider subnets to be local.

4.2.6 TCP sliding window

TCP enforces flow control of data from the sender to the receiver through a mechanism referred to as *sliding window*. This helps ensure delivery to a receiving application. The size of the window is defined by the tcp_sendspace and tcp_recvspace values.

The window is the maximum amount of data that a sender can send without receiving any ACK segment. A receiver always advertises its window size in the TCP header of the ACK segments.

In the example in Figure 7 on page 37, the sending application is sleeping because it has attempted to write data that would cause TCP to exceed the send socket buffer space (that is, tcp_sendspace). The sending TCP has sent the last part of rec5, all of rec6 and rec7, and the beginning of rec8. The receiving TCP has not yet received the last part of rec7 or any of rec8.

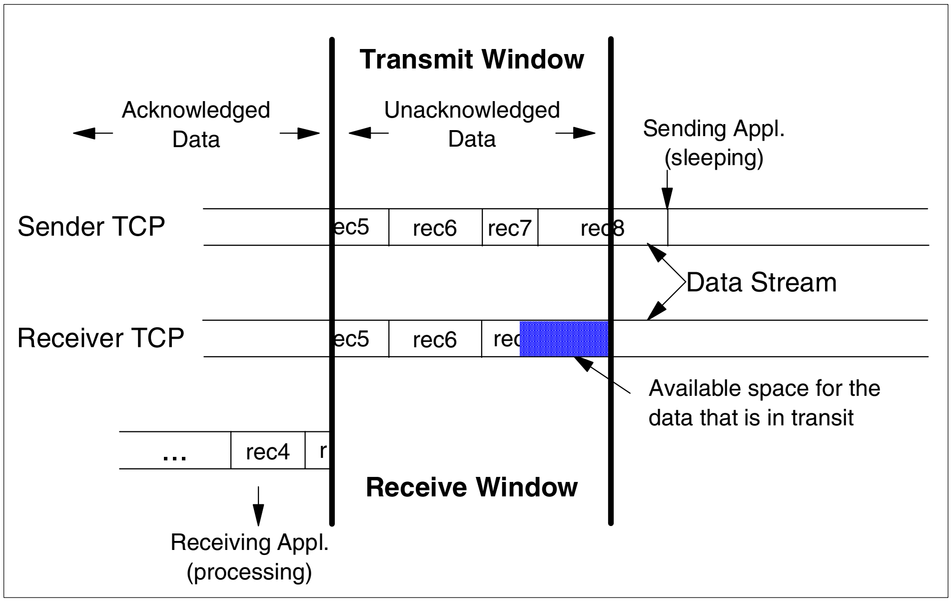

Figure 7. TCP Sliding Window

The receiving application got rec4 and the beginning of rec5 when it last read the socket, and it is now processing that data. When the receiving application next reads the socket, it will receive (assuming a large enough read) the rest of rec5, rec6, and as much of rec7 and rec8 as has arrived by that time.

In the course of establishing a session, the initiator and the listener converse to determine their respective capacities for buffering input and output data. The smaller of the two sizes defines the size of the effective window. As data is written to the socket, it is moved into the sender's buffer. When the receiver indicates that it has space available, the sender transmits enough data to fill that space (assuming that it has that much data). It then informs the sender that the data has been successfully delivered. Only then does the sender discard the data from its own buffer, effectively moving the window to the right by the amount of data delivered. If the window is full because the receiving application has fallen behind, the sending thread will be blocked.

We now have a lot of high-speed network media and memory for a workstation. The maximum of 64 KB for a window may not be big enough for such an advanced environment. TCP has been enhanced to support such situations by *RFC 1323, TCP Extensions for High Performance*.

If the *rfc1323* parameter is 1, the maximum TCP window size is 4 GB (instead of 64 KB). Figure 8 illustrates this TCP enhancement.

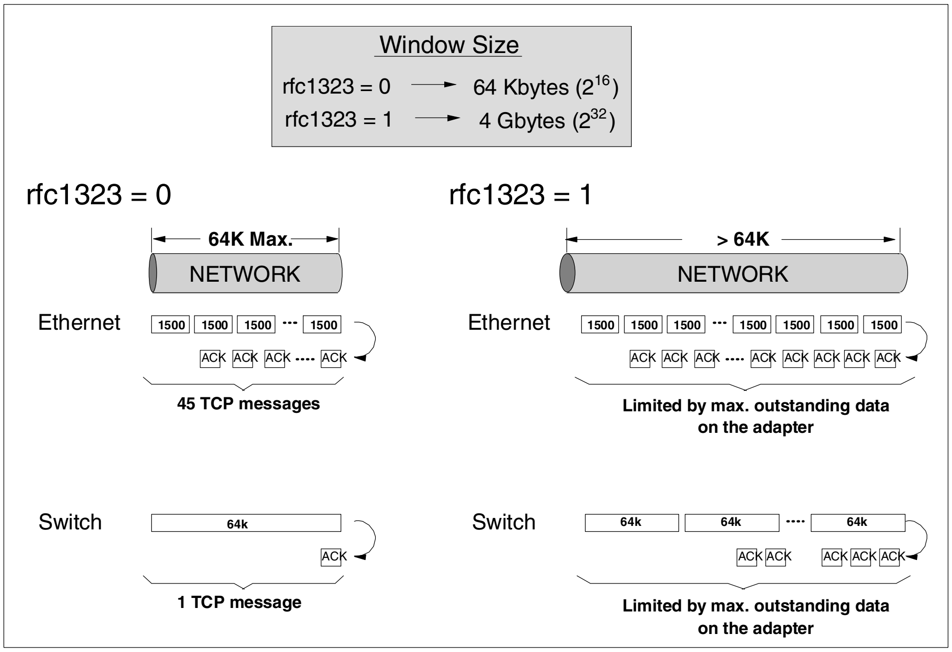

Figure 8. rfc1323 - TCP extension

There is, of course, no such thing as free function. The additional operations performed by TCP to ensure a reliable connection result in about 7 to 12 percent higher CPU time cost than in UDP.

There are two technical terms about TCP windows that you may sometimes get confused. A window is a receiver's matter, telling how much data the receiver can accept. There is also the term *send window*, which is a sender's matter. They are the same thing, but on some occasions when the congestion avoidance algorithm is working (see the following paragraphs), they represent different values to each other.

Also, many improvements have been made to the TCP sliding window mechanism. Here is a brief list of these improvements in RFC 1122 “Requirements for Internet Hosts - Communication Layers:”

- Silly window syndrome avoidance algorithm

  The Silly Window Syndrome (SWS) is caused when a large amount of data is transmitted. If the sender and receiver do not implement this algorithm, the receiver advertises a small amount of the window each time the receiver buffer is read by an application. As a result, the sender has to send a lot of small segments, which do not have the advantage of bulk data transfer; that is the purpose of the window mechanism. This algorithm is mandatory by RFC 1122.

- Delayed ACK

  TCP should not send back an ACK segment immediately because there will not be any data that can be sent with the ACK. As a result, the network traffic and protocol module overhead will be reduced. With this mechanism, an ACK segment is not returned immediately. This mechanism is optional (but strongly recommended) by RFC 1122.

- Nagle Algorithm

  When an application issues many write system calls with a single byte of data or so, TCP should not send data segments just carrying a single byte of data. In order to avoid this inefficiency, data should not be sent until the ACK of the prior data segment is received. This mechanism accumulates small segments into one big segment before it is sent out. This mechanism is optional by RFC 1122.

Note

Certain applications, such as X-Windows, do not work well with this mechanism. Thus, there must be an option to disable this feature. For this, you can use the TCP_NODELAY option of the setsockopt() call. For more information about the Nagle Algorithm, see Section 4.2.9, “Description of the Nagle Algorithm” on page 42.

- Congestion avoidance

  When a segment is lost, the sender's TCP module considers that this is due to the congestion of the network and reduces the send window size by a factor of 2. If the segment loss continues, the sender's TCP module keeps reducing the send window using the previous procedure until it reaches 1. This mechanism is mandatory by RFC 1122.

- Slow start

  If network congestion is resolved, the minimized send window should be recovered. The recovery should not be the opposite of shrinking (exponential backoff). This mechanism defines how to recover the send window. It is mandatory by RFC 1122.

4.2.7 Address resolution

The first network-level protocol is the Address Resolution Protocol (ARP). ARP dynamically translates Internet addresses for IP into unique hardware MAC addresses. The kernel maintains the translation tables, and the ARP is not directly available to users or applications. When an application sends an Internet packet to one of the interface drivers, the driver requests the appropriate address mapping. If the mapping is not in the table, an ARP broadcast packet is sent through the requesting interface driver to the hosts on the local area network.

Entries in the ARP mapping table are deleted after 20 minutes; incomplete entries are deleted after 3 minutes. This value can be changed with the `no` command option arpt_killc.

When any host that supports ARP receives an ARP request packet, the host notes the IP and hardware addresses of the requesting system and updates its mapping table (if necessary). If the receiving host IP address does not match the requested address, the host discards the request packet. If the IP address does match, the receiving host sends a response packet to the requesting system. The requesting system stores the new mapping and uses it to transmit any similar pending Internet packets into the unique hardware addresses on local area networks.

4.2.7.1 ARP cache tuning

The relevant `no` command options for tuning ARP cache are shown in Table 4:

Table 4. Default ARP parameters in AIX

Parameter	AIX Default Value	Definition
arptab_nb	25	Number of buckets.
arptab_bsiz	7	Number of entries in each bucket.
arpqsize	1	Number of packets to queue while waiting for ARP responses.

Calculating ARP entries

The total available ARP entries are calculated using the variables:

arptab_nb * arptab_bsiz.

In a default configuration, this gives us 175 ARP entries.

arpqsize is a runtime attribute, but to change the others, it is necessary to include these lines at the beginning of the /etc/rc.net script, right after the first line in the file, as shown in Figure 9:

```
#!/bin/ksh
no -o arptab_nb=64
no -o arptab_bsiz=8
# IBM_PROLOG_BEGIN_TAG
```

Figure 9. Inserting no options for ARP cache in /etc/rc.net.

A complete description of these `no` command options can be found in Section 4.3.2, “AIX tunables” on page 46.

As general recommendations for ARP cache sizing, you can use these:

- For fast lookups, a large number of small buckets is ideal.
- For memory efficiency, a small number of medium buckets is best. Having too many buckets wastes memory (if the arptab_nb size were set to 128, bucket numbers above 66 would rarely be used).

You can also evaluate your current parameters. Use `arp -a` to get the current contents of your ARP cache. See if any of your buckets are full. You can do this by pinging an IP address on a local subnet that is not in the ARP cache and is not being used. See how long the ARP entry stays in the cache. If it lasts for a few minutes, that particular bucket is not a problem. If it disappears quickly, that bucket is doing some thrashing. Carefully choosing the IP addresses to ping will let you monitor different buckets. Make sure the ping actually made the round trip before timing the ARP cache entry.

ARP cache thrashing, generally, shows us the ARP cache containing a large number of entries.

In Section 9.3, “ARP cache tuning” on page 193, you can find the suggested way to tune `no` command options for ARP cache in SP systems.

4.2.8 Subnet addressing

Subnet addressing allows an autonomous system made up of multiple networks to share the same Internet address. The subnetwork capability of TCP/IP also makes it possible to divide a single network into multiple logical networks (subnets). For example, an organization can have a single Internet network address that is known to users outside the organization, yet configure its network internally into departmental subnets. In either case, fewer Internet network addresses are required while local routing capabilities are enhanced.

A standard Internet Protocol address field has two parts: a network address and a local address. To make subnets possible, the local address part of an Internet address is divided into a subnet number and a host number. The subnet is identified so that the local autonomous system can route messages reliably.

4.2.9 Description of the Nagle Algorithm

The Nagle Algorithm (RFC 896) was designed for slower networks to encourage the use of large packets rather than many small ones; to avoid excessive LAN traffic when an application is reading in small IP packets, by delaying the TCP acknowledgement (ACK) until the receiving application has read a total amount of data that is at least half the receive window size or twice the maximum segment size.

The Nagle Algorithm is particularly helpful for networks such as Ethernet, which behave badly with many small packets due to its collision avoidance algorithm. However, it can cause problems on large segment size networks, such as SP Switch, HiPPI or ATM.

TCP_NODELAY specifies whether TCP should follow the Nagle Algorithm for deciding when to send data. By default, TCP will follow it. To disable this behavior, applications can enable TCP_NODELAY to force TCP to always send data immediately.

Nagle Algorithm

The Nagle Algorithm states that under some circumstances, there will be a waiting period of 200 msec before data is sent. The Nagle Algorithm uses the following parameters for traffic over a switch:

- Segment size = MTU or tcp_mssdflt or MTU path discovery value
- TCP window size = smaller of tcp_sendspace and tcp_recvspace values
- Data size = application data buffer size

The specific rules used by the Nagle Algorithm for deciding when to send data follows:

- If a packet is equal to or larger than the segment size (or MTU), send it immediately.
- If the interface is idle, or the TCP_NODELAY flag is set, and the TCP window is not full, send the packet immediately.

- If there is less than half of the TCP window in outstanding data, send the packet immediately.
- If sending less than a segment size, and if more than half the window is outstanding, and TCP_NODELAY is not set, wait up to 200 msec for more data before sending the packet.

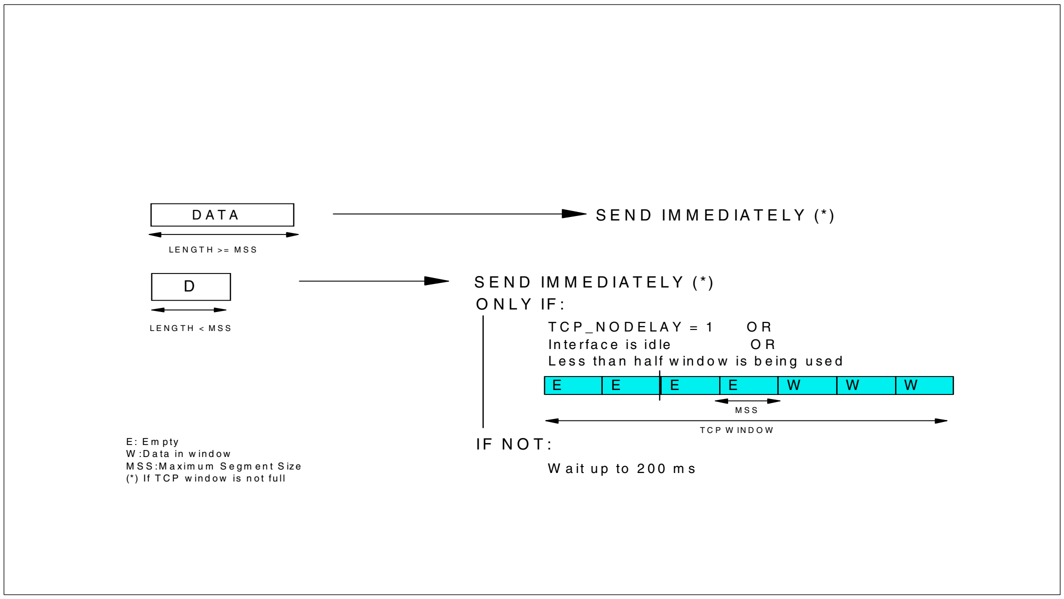

Figure 10. The Nagle Algorithm

In addition to the Nagle Algorithm, a delayed ACK timer can cause slow data transfer over an SP switch. This timer is set when a single TCP/IP packet is received at the receive side of a connection. If a second packet is received, then the timer expires, and an acknowledgement is sent back to the sending side. You rarely see this on small segment size (MTU) networks because a large buffer of data results in more than one packet being transmitted. However, on large segment size networks like the SP Switch, writing a 32 KB buffer results in only one packet. That same buffer on smaller segment size networks results in multiple packets, and the delay ACK timer is not used.

With the *rfc1323* parameter not set to 1, and having a large segment size for the network, sending full IP packets can cause a 5 packet/second rate. Table 5 lists the window size where this occurs for various network segment sizes.

Table 5. TCP/IP Pacing Degradation Window

Network type	MTU	TCP window Nagle hole
Ethernet	1500	1501-2999
Token Ring	1492	1493-2983
Token Ring	4096	4097-8191
FDDI	4352	4353-8705
ATM	9180	9181-18359
ATM	60416	60417-120831
SP Switch	65520	65521-131039
FCS	65536	65537-131071
HiPPI	65536	65537-131071

The effect of the Nagle Algorithm or delayed ACK timer is easy to see if only one socket connection is running. If you check the packet rate over the switch, you should see an average of 5 packets/sec. Typically, a transfer rate of 150 to 300 KB/second is reported by an application. To check the packet rate over the switch, use the following command:

```
# netstat -I css0 1
```

The output will show the switch and total IP packet rates per second, as shown in Figure 11.

```
# netstat -I css0 1
    input   (css0)     output            input   (Total)    output
 packets  errs  packets  errs colls  packets  errs  packets  errs colls
  125696     0   110803     0     0   356878     0   287880     0     0
     119     0      216     0     0      123     0      221     0     0
     117     0      222     0     0      120     0      224     0     0
     115     0      225     0     0      117     0      227     0     0
     115     0      202     0     0      117     0      204     0     0
     115     0      207     0     0      117     0      209     0     0
     116     0      201     0     0      118     0      203     0     0
```

Figure 11. netstat -I command

In Section 9.1, “The Nagle Algorithm” on page 191, you can find clues to detect this problem in SP systems and some recommendations to avoid it.

4.2.10 Routing

In TCP/IP, routing can be one of two types: static or dynamic. With static routing, you maintain the routing table manually using the `route` command. Static routing is practical for a single network communicating with one or two other networks. However, as your network begins to communicate with more networks, the number of gateways increases, and so does the amount of time and effort required to maintain the routing table manually.

4.3 AIX network tunables

This section describe the network parameters involved in the process of tuning an SP system. These parameters include the `no` command options and `nfso` command options.

Although there are 95 `no` parameters that you can use, we only explain the most relevant. All default values are referred to AIX 4.3.3. We also discuss the `nfso` parameters related to NFS tuning in SP systems. You can find a complete information about `no` values and `nfso` values in Understanding IBM RS/6000 Performance and Sizing, SG24-4810.

The tunable values should be set to customized values during the installation process. See Section 4.7, “Tuning the SP network for specific workloads” on page 86 for recommended default settings. Use the `no` and `nfso` commands to display the current settings.

IPv6

SP switch adapter only supports IPv4 addressing, that is, it can talk to IPv6 networks because of a mapping method, but does not support IPv6 addressing.

We do not include any IPv6 parameters in this chapter for that reason.

4.3.1 IP tunable inheritance rules

This is the list of rules that apply for IP tunables inheritance. All parameters described in the next section follow them.

- All socket connections inherit values at create time from `no` option settings.
- If Interface Specific Network Options (ISNO) are implemented, use the network setting for the adapter being used.
- Changes to `no` command do not affect existing socket connections.

- Child processes inherit default socket settings from STDIN and STDOUT from parent processes.
- Processes spawned by inetd, inherit `no` tunable settings for STDIN and STDOUT from the time when inetd was started.
- Application specific settings set using setsockopt() system call always override the defaults.

4.3.2 AIX tunables

For each parameter listed we describe the purpose, the default value, the way to change/show the current value and the suggested one for use in an SP system. All default values are referred to AIX 4.3.3. Most of them are managed with the `no` command.

`no`

Purpose: This is the command used to configure network attributes.

Syntax: `no { -a | -d option | -o option [=newvalue]}`

Note

Be careful when you use this command. The `no` command performs no range checking; therefore, it will accept all values for the variable. If used incorrectly, the `no` command can make your system inoperable.

4.3.2.1 Memory-related tunables

The following tunable parameters are related to communication memory management, for example, related to the way that the AIX operating system handles the memory used by the communication subsystem.

thewall

Purpose: This specifies the maximum amount of memory, in KB, that is allocated to the memory pool. thewall is a runtime attribute. Systems that are not Common Hardware Reference Platform (CHRP), that is, Power PC based, are limited to 256 MB.

AIX 4.2.1 and earlier: smaller of 1/8 RAM or 64MB

AIX 4.3.0: smaller of 1/8 RAM or 128 MB

AIX 4.3.1: smaller of 1/2 RAM or 128 MB

AIX 4.3.2 and later: smaller of 1/2 RAM or 1GB (CHRP)

Values: Defaults: see "Purpose"

Display: `no -o thewall`

Change: `no -o thewall=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: Increase size, preferably to multiples of 4 KB.

> **Note**
>
> In AIX version 4.3.2 and later, this value should not need to be manually set. In most cases, the system will automatically select a setting that will be appropriate. However, if network buffer errors occur, it may need to be manually changed.

maxmbuf

Purpose: This is also used to limit how much real memory can be used by the communications subsystem. Prior to AIX Version 4.2 the upper limits on mbufs is the higher value of maxmbuf or thewall. In AIX 4.2. and later, if it is greater than 0 the maxmbuf value is used regardless of the value of thewall. The value of 0 indicates that the system default (thewall) should be used.

Values: Default: 0

Display: `lsattr -E -l sys0`

Change: `chdev -l sys0 -a maxmbuf=newvalue`

Changes take effect immediately and are permanent.

sb_max

Purpose: This provides an absolute upper bound on the size of TCP and UDP socket buffers per socket. It limits setsockopt(), udp_sendspace, udp_recvspace, tcp_sendspace and tcp_recvspace variables. The units are in bytes.

Values: Default: 1048576

Display: `no -o sb_max`

Change: `no -o sb_max=newvalue`

Changes take effect immediately for new connections and are effective until next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: Increase size, preferably to multiples of 4096. Should be at least twice the size of the largest value for tcp_sendspace, tcp_recvspace, udp_sendspace and udp_recvspace. This ensures that if the buffer utilization is better than 50 percent efficient, the entire size of the tcp and udp byte limits can be used.

extendednetstats

Purpose: This enables more extensive statistics for network memory services.

Values: Default: 1 (in /etc/rc.net is set to 0). Range: 0 or 1

Display: `no -o extendednetstats`

Change: `no -o extendednetstats=newvalue`

Changes take effect immediately for new connections and are effective until next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: When it is set to 1, these extra statistics cause a reduction in system performance. If, for tuning reasons, these statistics are desired, it is recommended that the code in /etc/rc.net that sets extendednetstats to 0 be commented out.

use_isno

Purpose: This enables or disables per interface tuning options. Only applies to version 4.3.3. There are five parameters that can be tuned for each supported network interface:

- rfc1323
- tcp_nodelay
- tcp_sendspace
- tcp_recvspace
- tcp_mssdflt

These are available for all of the mainstream TCP/IP interfaces: Token-Ring, 10/100 Ethernet, Gigabit Ethernet, FDDI and ATM (a software update is required). It is not supported in SP Switch or SP Switch2 interface, but, as a workaround, SP Switch users can set the tuning options appropriate for the switch using the traditional `no` command and then use ISNO to override the values needed for the other system interfaces.

Values: Default: 1(yes). Range: 0 or 1.

Display: `no -o use_isno`

Change: `no -o use_isno=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: If the TCP tunable parameters per interface (tunable through `smit` or `chdev` command) have been set, they will override the TCP global values if use_isno is set on. Applications can still override all of these with the setsockopt() subroutine.

Table 6 lists the parameters for the setsockopt() subroutine.

Table 6. Socket level network option tunables

setsockopt()	no option equivalent
SO_SNDBUF	tcp_sendspace
SO_RCVBUF	tcp_recvspace
RFC1323	rfc1323
TCP_NODELAY	n/a (use `lsattr -El` interface)
n/a	sb_max
n/a	tcp_mssdflt

tcp_sendspace

Purpose: This provides the default value of the size of the TCP socket send buffer, in bytes.

Values: Default: 4096 bytes (in /etc/rc.net is set to 16384).

Range: 0 to 64 KB, if rfc1323=0.

Range: 0 to 4 GB, if rfc1323=1. In any case, it should be less or equal than sb_max.

Display: `no -o tcp_sendspace`

ISNO: `lsattr -El interface` or `ifconfig interface`

Change: `no -o tcp_sendspace=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

ISNO:

`ifconfig interface tcp_sendspace newvalue` or `chdev -l interface -a tcp_sendspace=newvalue`

The `ifconfig` command sets values temporarily, making it useful for testing. The `chdev` command alters the ODM, so custom values return after system reboots.

Tuning: This affects the window size used by TCP. Setting the socket buffer size to 16 KB improves performance over standard Ethernet and Token-Ring networks. The optimum buffer size is the product of the media bandwidth and the average round-trip time of a packet.

The tcp_sendspace parameter is a runtime attribute, but for daemons started by the inetd daemon, run the following commands:

`stopsrc -s inetd; startsrc -s inetd`

In SP systems, set a minimum of 65536 unless there are problems with external connections not handling greater than 32768 byte windows. Never set it higher than the major network adapter transmit queue limit. To calculate this limit, use:

`major adapter queue size * major network adapter MTU`

tcp_recvspace

Purpose: This provides the default value of the size of the TCP socket receive buffer, in bytes.

Values: Default: 4096 bytes (in /etc/rc.net is set to 16384).

Range: 0 to 64 KB, if rfc1323=0.

Range: 0 to 4 GB, if rfc1323=1. In any case, it should be less or equal than sb_max.

Display: `no -o tcp_recvspace`

ISNO: `lsattr -El interface` or `ifconfig interface`

Change: `no -o tcp_recvspace=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

ISNO:

`ifconfig interface tcp_recvspace newvalue` or

`chdev -l interface -a tcp_recvspace=newvalue`

The `ifconfig` command sets values temporarily, making it useful for testing. The `chdev` command alters the ODM, so custom values return after system reboots.

Tuning: This affects the window size used by TCP. Setting the socket buffer size to 16 KB improves performance over standard Ethernet and Token-Ring networks. The optimum buffer size is the product of the media bandwidth and the average round-trip time of a packet. The tcp_recvspace parameter is a runtime attribute, but for daemons started by the inetd daemon, run the following commands:

```
stopsrc -s inetd; startsrc -s inetd
```

In SP systems, set a minimum of 65536, unless there are problems with external connections not handling greater than 32768 byte windows. Never set higher than the major network adapter transmit queue limit. To calculate this limit, use:

```
major adapter queue size * major network adapter MTU
```

Note

To properly set these tunables, you need a good understanding of the type of traffic your application will be sending. For optimal switch performance on a single socket, tcp_sendspace and tcp_recvspace need to be set to 384 KB or greater.

udp_sendspace

Purpose: This provides the default value of the size of the UDP socket send buffer, in bytes.

Values: Default: 9216, Range: 0 to 65536

Display: `no -o udp_sendspace`

Change: `no -o udp_sendspace=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: Increase size, preferably to a multiple of 4096. This should always be less than udp_recvspace and less than or equal to sb_max, but never greater than 65536.

udp_recvspace

Purpose: This provides the default value of the size of the UDP socket receive buffer.

Values: Default: 41600

Display: `no -o udp_recvspace`

Change: `no -o udp_recvspace=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: Increase size, preferably to a multiple of 4096.This should always be greater than udp_sendspace and less than or equal to sb_max. Adjust the values to handle as many simultaneous UDP packets as can be expected per UDP socket.

Diagnosis:

```
# netstat -s
udp:
       n socket buffer overflows.
```

with n greater than 0.

In SP systems, a good suggestion for a starting value is 10 times the value of udp_sendspace, because UDP may not be able to pass a packet to the application before another one arrives. Also, several nodes can send to one node at the same time. To provide some staging space, this size is set to allow 10 packets to be staged before subsequent packets are thrown away. For large parallel applications using UDP, the value may have to be increased.

rfc1323

Purpose: This enables TCP enhancements as specified by RFC 1323 (TCP extensions for High Performance).Value of 1 indicates that tcp_sendspace and tcp_recvspace sizes can exceed 64 KB.

If the value is 0, the effective tcp_sendspace and tcp_recvspace sizes are limited to a maximum of 65535.

Values: Default: 0, Range: 0 or 1

Display: `no -o rfc1323`

ISNO: `lsattr -El interface` or `ifconfig interface`

Change: `no -o rfc1323=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

ISNO:

`ifconfig interface rfc1323 newvalue` or

`chdev -l interface -a rfc1323=newvalue`

The `ifconfig` command sets values temporarily, making it useful for testing. The `chdev` command alters the ODM, so custom values return after system reboots.

Tuning: The sockets application can override the default behavior on individual connections using the setsockopt() subroutine.

In SP systems always set in each side of the connection to 1.

4.3.2.2 Networking-related tunables

The following tunable parameters are related to networking management, that is, related to the communications and negotiations between the parts of a network.

subnetsarelocal

Purpose: This specifies that all subnets that match the subnet mask are to be considered local for purposes of establishing, for example, the TCP maximum segment size (instead of using MTU).This attribute is used by the in_localaddress() subroutine. The value of 1 specifies that addresses that match the local network mask are local. If the value is 0, only addresses matching the local subnetwork are local. It is a runtime attribute.

Values: Default: 1 (yes), Range: 0 or 1

Display: `no -o subnetsarelocal`

Change: `no -o subnetsarelocal=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: This is a configuration decision with performance consequences. If the subnets do not all have the same MTU, fragmentation at bridges may degrade performance. If the subnets do have the same MTU, and subnets that are local are 0, TCP sessions may use an unnecessarily small MSS.

In SP systems, always set to 1.

ipforwarding

Purpose: This specifies whether the kernel should forward IP packets.

Values: Default: 0 (no), Range: 0 or 1

Display: `no -o ipforwarding`

Change: `no -o ipforwarding=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Comment: This is a configuration decision with performance consequences.

In SP systems, set to 1 for gateway nodes (or nodes that really need it).

tcp_mssdflt

Purpose: This specifies the default maximum segment size used in communicating with remote networks. However, only one value can be set, even though there are several adapters with different MTU sizes. It is the same as MTU for communication across a local network, except for one small difference: the tcp_mssdflt size is for the size of only the data in the packet. You need to reduce the value for the size of any headers so that you can send full packets instead of a full packet and a fragment.

Values: Default: 512, Range: 512 to unlimited

Display: `no -o tcp_mssdflt`

ISNO: `lsattr -El interface` or `ifconfig interface`

Change: `no -o tcp_mssdflt=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

ISNO:

`ifconfig interface tcp_mssdflt newvalue` or

`chdev -l interface -a tcp_mssdflt=newvalue`

The `ifconfig` command sets values temporarily, making it useful for testing. The `chdev` command alters the ODM, so custom values return after system reboots.

Tuning: In AIX 4.2.1 or later, tcp_mssdflt is only used if path MTU discovery is not enabled or path MTU discovery fails to discover a path MTU. Limiting data to (MTU - 52) bytes ensures that, where possible, only full packets will be sent. If set higher than MTU of the adapter, the IP or intermediate router fragments packets. The way to calculate this is as follows:

MTU of interface - TCP header size - IP header size - rfc1323 header size

which is:

MTU - (20 + 20 + 12), or MTU - 52, if rfc1323 = 1

MTU - (20 + 20), or MTU - 40, if rfc1323 = 0

In SP systems set to 1448.

tcp_nodelay

Purpose: This is not a `no` parameter, but is related in terms of its effect on performance. Specifies that sockets using TCP over this interface follow the Nagle Algorithm when sending data. By default, TCP follows the Nagle Algorithm. See Section 4.2.9, “Description of the Nagle Algorithm” on page 42 for more details.

Values: Default: 0, Range: 0 or 1

Display: `lsattr -E -l interface` or `ifconfig interface`

Change: `chdev -l interface -a tcp_nodelay=newvalue`

or

`ifconfig interface tcp_delay newvalue`

The `ifconfig` command sets values temporarily, making it useful for testing. The `chdev` command alters the ODM, so custom values return after system reboots.

MTU

Purpose: This is not a `no` parameter, but is related in terms of its effect on performance. Limits the size of the packets that are transmitted on the network, in bytes.

Values: Default: adapter dependent. Range: 512 bytes to 65536 bytes

Display: `lsattr -E -l interface`

Change: `chdev -l interface -a mtu=newvalue`

Because all the systems on the LAN must have the same MTU, they must change simultaneously. With the `chdev` command, the interface cannot be changed while it is in use. Change is effective across boots. An alternative method is as follows:

`ifconfig interface mtu newvalue`

This changes the MTU size on a running system, but will not preserve the value across a system reboot.

Tuning: The default size should be kept.

In SP systems, the CSS MTU size is 65520. We suggest that you keep this value and always use the same MTU across all nodes in the SP.

Note

Under some circumstances, for example, when you want to avoid the Nagle Algorithm, causing very slow traffic, it may be necessary to reduce the CSS MTU size. You can reduce the value for a switch to 32678 with only a two to ten percent loss in throughput. But CPU utilization will be slightly higher due to the per-packet overhead.

In Table 3 on page 29, you can find a detailed table with MTU values for different network adapters.

tcp_pmtu_discover

Purpose: This enables or disables path MTU discovery for TCP applications.

Values: Default: 1 (yes). Range: 0 or 1. In earlier versions than AIX 4.3.3 the default is 0.

Display: `no -o tcp_pmtu_discover`

Change: `no -o tcp_pmtu_discover=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: Watch out for networks with more than a couple of hundred hosted. If you turn tcp_pmtu_discover on, you are, in effect, creating a route in your routing table to every host that is out there. Any network greater than a couple of hundred hosts becomes very inefficient and performance problems arise.

udp_pmtu_discover

Purpose: This enables or disables path MTU discovery for UDP applications.

Values: Default: 1 (yes). Range: 0 or 1. In earlier versions than AIX 4.3.3 the default is 0.

Display: `no -o udp_pmtu_discover`

Change: `no -o udp_pmtu_discover=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: UDP applications must be specifically written to use path MTU discovery.

ipqmaxlen

Purpose: This specifies the number of received packets that can be queued on the IP protocol input queue. It is a load-time attribute and must be set in /etc/rc.net file prior to the interfaces being defined.

Values: Default: 100.

Display: `no -o ipqmaxlen`

Change: `no -o ipqmaxlen=newvalue`

Tuning: Increase this value if system is using a lot of loopback sessions.

Diagnosis:

```
# netstat -s
ip:
       n ipintrp overflows.
```

with n greater than 0.

In SP systems, increase this value to 512.

4.3.2.3 ARP cache-related tunables

The following tunable parameters are related to the ARP cache and its sizing.

arptab_nb

Purpose: This specifies the number of arp table buckets.

Values: Default: 25.

Display: `no -o arptab_nb`

Change: `no -o arptab_nb=newvalue`

Change is effective when the netinet kernel extension is loaded. It is a load-time attribute and must be set in /etc/rc.net file prior to the interfaces being defined. See Figure 9 on page 41 for more details.

Tuning: Increase this value for systems that have a large number of clients or servers. The default provides for 25 x 7 = 175 arp entries, but assumes an even task distribution.

arptab_bsiz

Purpose: This specifies the arp table bucket size.

Values: Default: 7.

Display: `no -o arptab_bsiz`

Change: `no -o arptab_bsiz=newvalue`

Change is effective when the netinet kernel extension is loaded. It is a load-time attribute and must be set in /etc/rc.net file prior to the interfaces being defined. See Figure 9 on page 41 for more details.

Tuning: Increase this value for systems that have a large number of clients or servers. The default provides for 25 x 7 = 175 arp entries, but assumes an even task distribution.

For sizing ARP cache in SP systems, you can follow the recommended values included in Table 7.

Table 7. Recommended values for ARP cache sizing in SP systems

Number of nodes	arptab_nb	Number of interfaces	arptab_bsiz
1 - 64	25	1 - 3	7
65 - 128	64	4	8

Number of nodes	arptab_nb	Number of interfaces	arptab_bsiz
129 - 256	128	5	10
257 - 512	256	more ...	2 x # of interfaces

arpqsize

Purpose: This specifies the maximum number of packets to queue while waiting for arp responses. It is supported by Ethernet, 802.3, Token-Ring and FDDI interfaces. This is a runtime attribute.

Values: Default: 1.

Display: `no -o arpqsize`

Change: `no -o arpqsize=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: The arpqsize value is increased to a minimum value of 5 when path MTU discovery is enabled. The value will not automatically decrease if path MTP discovery is subsequently disabled.

arpt_killc

Purpose: This specifies the time (in minutes) before a complete ARP entry will be deleted. This is a runtime attribute.

Values: Default: 20.

Display: `no -o arpt_killc`

Change: `no -o arpt_killc=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: To reduce ARP activity in a stable network, you can increase arpt_killc.

ifsize

Purpose: This specifies the maximum number of network interface structures per interface.

Values: Default: 8.

Display: `no -o ifsize`

Change: `no -o ifsize=newvalue`

Change is effective when the netinet kernel extension is loaded. It is a load-time attribute and must be set in /etc/rc.net file prior to the interfaces being defined. See Figure 9 on page 41 for more details.

Tuning: In AIX 4.3.2 and above, if the system detects, at boot time, that more adapters of one type are present than it would be allowed by the current value of ifsize, it will automatically increase the value to support the number of adapters present.

4.3.2.4 NBC-related tunables

The following tunable parameters are related to the Network Buffer Cache (NBC) and its sizing. The NBC is a list of network buffers which contains data objects that can be transmitted to the networks. It grows dynamically as data objects are added or removed from it. The Network Buffer Cache is used by some kernel interfaces for performance enhancement on the network.

In AIX 4.3.2 and later, you can see the NBC statistics:

```
# netstat -c
Network Buffer Cache Statistics:
-------------------------------
Current total cache buffer size: 0
Maximum total cache buffer size: 0
Current total cache data size: 0
Maximum total cache data size: 0
Current number of cache: 0
Maximum number of cache: 0
Number of cache with data: 0
Number of searches in cache: 0
Number of cache hit: 0
Number of cache miss: 0
Number of cache newly added: 0
Number of cache updated: 0
Number of cache removed: 0
Number of successful cache accesses: 0
Number of unsuccessful cache accesses: 0
Number of cache validation: 0
Current total cache data size in private segments: 0
Maximum total cache data size in private segments: 0
Current total number of private segments: 0
Maximum total number of private segments: 0
Current number of free private segments: 0
Current total NBC_NAMED_FILE entries: 0
Maximum total NBC_NAMED_FILE entries: 0
#
```

nbc_limit

Purpose: This specifies the total maximum amount of memory that can be used for the Network Buffer Cache, in KB.

Values: Default: derived from thewall parameter.

Display: `no -o nbc_limit`

Change: `no -o nbc_limit=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: When the cache grows to this limit, the least-used caches are flushed out of cache to make room for the new ones. This attribute only applies in AIX 4.3.2 and later. NBC is only used by the send_file() API and some web servers that use the get engine in the kernel.

nbc_max_cache

Purpose: This specifies the maximum size of the cache object allowed in the Network Buffer Cache (NBC), in bytes, without using the private segments.

Values: Default: 131072 (128 KB).

Display: `no -o nbc_max_cache`

Change: `no -o nbc_max_cache=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: Data objects bigger than this size are either cached in a private segment or are not cached at all, but may not be put in the NBC. This attribute only applies in AIX 4.3.2 and later. NBC is only used by the send_file() API and some web servers that use the get engine in the kernel.

nbc_min_cache

Purpose: This specifies the minimum size of the cache object allowed in the Network Buffer Cache (NBC), in bytes, without using the private segments.

Values: Default: 1 byte.

Display: `no -o nbc_min_cache`

Change: `no -o nbc_min_cache=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: Data objects smaller than this size are either cached in a private segment or are not cached at all, but may not be put in the NBC. This attribute only applies in AIX 4.3.2 and later. NBC is only used by the send_file() API and some web servers that use the get engine in the kernel.

nbc_pseg

Purpose: This specifies the maximum number of private segments that can be created for the Network Buffer Cache (NBC).

Values: Default: 0.

Display: `no -o nbc_pseg`

Change: `no -o nbc_pseg=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: When this option is set to a nonzero value, data objects with size between the size specified in nbc_max_cache and the segment size (256 MB) will be cached in a private segment. Data objects bigger than the segment size will not be cached at all. When this many private segments exist in NBC, cache data in private segments may be flushed for new cache data so the number of private segments will not exceed the limit. When this option is set to 0, all cache in private segments will be flushed.

This attribute is new in AIX 4.3.3.

nbc_pseg_limit

Purpose: This specifies the maximum total cache data size allowed in private segments in the Network Buffer Cache (NBC), in KB.

Values: Default: half of real memory.

Display: `no -o nbc_pseg_limit`

Change: `no -o nbc_pseg_limit=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `no` command to /etc/rc.net.

Tuning: Because data cached in private segments will be pinned by the Network Buffer Cache, this option provides a control on the amount of pinned memory used for the Network Buffer Cache in addition to the network buffers in global segments. When this limit is met, cache data in private segments may be flushed for new cache data so the total pinned memory size will not exceed the limit. When this option is set to 0, all cache in private segments will be flushed.

This attribute is new in AIX 4.3.3.

4.3.2.5 NFS-related tunables

The following tunables parameters are related to Network File System (NFS). They are managed with the `nfso` command.

`nfso`

Purpose: This is the command used to configure Network File System (NFS) network variables.

Syntax: `nfso { -a | -d option | -l hostname | -o option [=newvalue]} [ -c ].`

Note

Be careful when you use this command. The `nfso` command performs no range checking; therefore, it accept all values for the variable. If used incorrectly, the `nfso` command can make your system inoperable.

biod

Purpose: This specifies the number of biod processes available to handle NFS requests on a client.

Values: Default: 6. Range: 1 to any positive integer

Display: `ps -efa |grep biod`

Change: `chnfs -b newvalue` - To change the value immediately and for each subsequent system reboot.

`chnfs -N -b newvalue` - To change the value immediately with no permanent change.

`chnfs -I -b newvalue` - To delay the change until next system reboot.

Tuning: Check `netstat -s` to look for UDP socket buffer overflows. Then, increase the number of biod daemons until socket buffer overflows cease. Refer to Section 4.5.3, “Tuning the numbers of nfsd and biod daemons” on page 72 for more details.

nfsd

Purpose: This specifies the number of nfsd processes available to handle NFS requests on a server.

Values: Default: 8. Range: 1 to any positive integer

Display: `ps -efa |grep nfsd`

Change: `chnfs -n newvalue` - To change the number immediately and for each subsequent system reboot.

`chnfs -N -n newvalue` - To change the number immediately with no permanent change.

`chnfs -I -n newvalue` - To delay the change until next system reboot.

Tuning: Check `netstat -s` to look for UDP socket buffer overflows. Then, increase the number of nfsd daemons until socket buffer overflows cease. Refer to Section 4.5.3, “Tuning the numbers of nfsd and biod daemons” on page 72 for more details.

nfs_socketsize

Purpose: This specifies the queue size of the NFS server UDP socket, in bytes. This socket is used for receiving the NFS client request.

Values: Default: 60000. Practical Range: 60000 to 204800

Display: `nfso -o nfs_socketsize`

Change: `nfso -o nfs_socketsize=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `nfso` command to /etc/rc.nfs. The position in the file is crucial. Add the entry immediately before the nfsd daemon is started and after the biod daemon is started.

Tuning: Increase the size of the nfs_socketsize variable when `netstat` reports packets dropped due to full socket buffers for UDP (and increasing the number of nfsd daemons has not helped).

It is recommended to at least double the socket buffer size to begin with when working with active servers.

nfs_tcp_socketsize

Purpose: This specifies the queue size of the NFS server TCP socket, in bytes. This socket is used for receiving the NFS client request.

Values: Default: 60000. Practical Range: 60000 to 204800

Display: `nfso -o nfs_tcp_socketsize`

Change: `nfso -o nfs_tcp_socketsize=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `nfso` command to /etc/rc.nfs. The position in the file is crucial. Add the entry immediately before the nfsd daemon is started and after the biod daemon is started.

Tuning: This option reserves, but does not allocate, memory for use by the send and receive socket buffers of the socket. Large or busy servers should have larger values until TCP NFS traffic shows no packets dropped from the output of the `netstat -s -p tcp` command.

It is recommended to at least double the socket buffer size to begin with when working with active servers.

nfs_device_specific_bufs

Purpose: This option allows the NFS server to use memory allocations from network devices, if the network device supports such a feature.

Values: Default: 1. Range: 0 or 1

Display: `nfso -o nfs_device_specific_bufs`

Change: `nfso -o nfs_device_specific_bufs=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `nfso` command to /etc/rc.nfs.

Tuning: Use of these special memory allocations by the NFS server can positively affect the overall performance of the NFS server. The default of 1 means the NFS server is allowed to use the special network device memory allocations. If the value of 0 is used, the NFS server will use the traditional memory allocations for its processing of NFS client requests. These are buffers managed by a network interface that result in improved performance (over regular mbufs) because no setup for DMA is required on these. Two adapters that support this include the Micro Channel ATM adapter and the SP Switch adapter.

In large SP system configurations, set this value to 1.

nfs_rfc1323

Purpose: This option enables the use of RFC1323 for NFS sockets. Use it to allow very large TCP window size negotiation (greater than 65535 bytes) between systems.

Values: Default: 0. Range: 0 or 1

Display: `nfso -o nfs_rfc1323`

Change: `nfso -o nfs_rfc1323=newvalue`

Change takes effect immediately and it is effective until the next system reboot. To make the change permanent, add the `nfso` command to /etc/rc.nfs.

Tuning: If using the TCP transport between NFS client and server, and both systems support it, this allows the systems to negotiate a TCP window size in a way that will allow more data to be “in-flight” between the client and server. This increases the potential throughput between client and server. Unlike the rfc1323 option of the `no` command, this only affects NFS and not other applications in the system. If the `no` command parameter rfc1323 is already set, this NFS option does not need to be set.

In large SP system configurations, set this value to 1.

4.4 SP system-specific network tunables

We will now describe the variables provided by PSSP to tune the size of the switch adapter pools. As an initial size, these variables should be twice the size of the largest of the tcp_sendspace, tcp_recvspace, udp_sendspace and udp_recvspace values. These pools reside in pinned kernel memory.

spoolsize

Purpose: This is the size of the SP Switch device driver send pool, in bytes.

Values: Default: 512 KB

Range: 512 KB to 16 MB for SP Switch

Range: 512 KB to 32 MB for SP Switch2

Display: `lsattr -E -l css0`

Change: `chgcss0 -l css0 -a spoolsize=newvalue`

Changes update the ODM, so these changes will be permanent. A reboot of the system is necessary in order to apply any changes.

rpoolsize

Purpose: This is the size of the SP Switch device driver receive pool in bytes.

Values: Default: 512 KB

Range: 512 KB to 16 MB for SP Switch

Range: 512 KB to 32 MB for SP Switch2

Display: `lsattr -E -l css0`

Change: `chgcss0 -l css0 -a rpoolsize=newvalue`

Changes update the ODM, so these changes will be permanent. A reboot of the system is necessary in order to apply any changes.

Note

When allocating the send pool and the receive pool, realize that this space is pinned kernel space in physical memory. This takes space away from other user applications and it is particularly important in small memory nodes.

4.5 NFS tuning

NFS operates on a client/server basis, that is, files that are physically resident on a server disk or other permanent storage are accessed through NFS on a client machine. For an illustration of this setup, see Figure 12. In this sense, NFS is a combination of a networking protocol, client and server daemons, and kernel extensions.

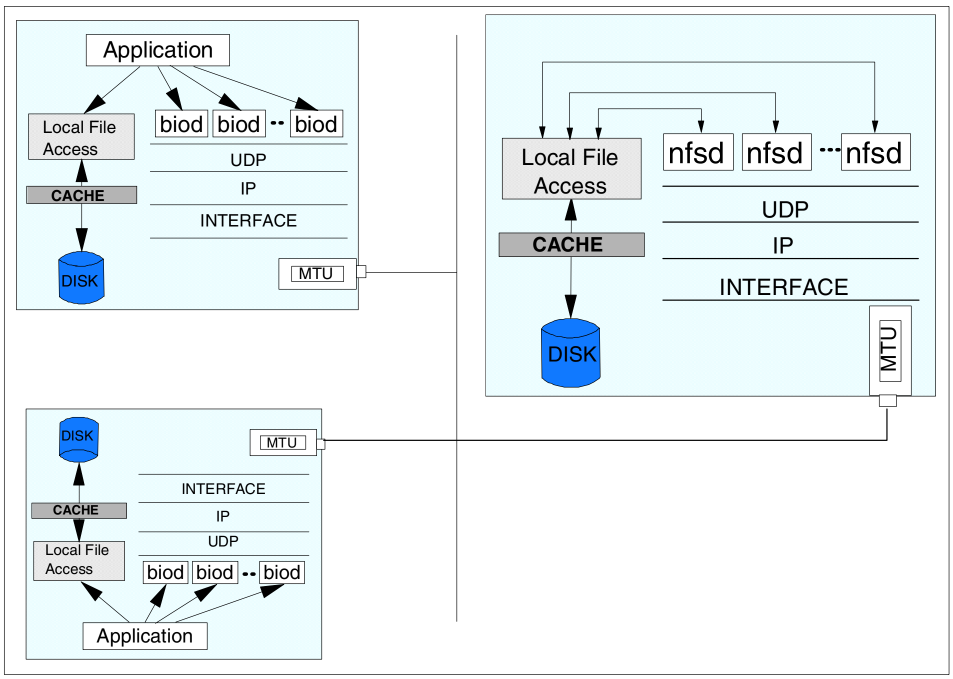

Figure 12. NFS overview

4.5.1 NFS overview for performance

mountd

The mountd daemon is a server daemon that answers a client request to mount a server's exported file system or directory. The mountd daemon determines which file system is available by reading the /etc/xtab file. Although soft-mounting the directories causes the error to be detected sooner, it runs a serious risk of data corruption. In general, read/write directories should be hard-mounted.

In the case of soft mounts, the RPC call will try some set number of times and then quit with an error. The default for NFS calls is three retransmits on a soft mount. For a hard mount, it will keep trying forever.

biod

The biod daemon is the block input/output daemon and is required in order to perform read-ahead and write-ahead requests, as well as directory reads. The biod daemon improves NFS performance by filling or emptying the buffer cache on behalf of the NFS clients. When a user on a client system wants to read or write to a file on a server, the biod daemon sends the requests to the server. Some NFS operations are sent directly to the server from the operating system's NFS client kernel extension and do not require the use of biod daemon.

nfsd

The nfsd daemon is the active agent providing NFS services from the NFS server. The receipt of any one NFS protocol request from a client requires the dedicated attention of an nfsd daemon until that request is satisfied and the results of the request processing are sent back to the client.

Mount process

The mount process takes place as follows:

1. Client mount makes call to server's portmap daemon to find the port number assigned to the mountd daemon.
2. The portmap daemon passes the port number to the client.
3. The client `mount` command then contacts the server mountd daemon directly and passes the name of the desired directory.
4. The server mountd daemon checks /etc/xtab (built by the `exportfs -a` command, which reads /etc/exports) to verify availability and permissions on the requested directory.

5. If all is verified, the server mountd daemon gets a file handle (pointer to file system directory) for the exported directory and passes it back to the client's kernel.

4.5.2 Read/write throughput

The following is an explanation of the read/write process in NFS and some suggestions to improve the throughputs on it.

4.5.2.1 Write throughput

Applications running on client systems may periodically write data to a file, changing the file's content. The amount of time an application waits for its data to be written to stable storage on the server is a measurement of the write throughput of a global file system. Write throughput is therefore an important aspect of performance. All global file systems, including NFS, must ensure that data is safely written to the destination file while at the same time minimizing the impact of server latency on write throughput.

The NFS Version 3 protocol increases write throughput by eliminating the synchronous write requirement of NFS Version 2 while retaining the benefits of close-to-open semantics. The NFS Version 3 client significantly reduces the number of write requests it makes to the server by collecting multiple requests and then writing the collective data through to the server's cache, but not necessarily to disk. Subsequently, NFS submits a commit request to the server that ensures that the server has written all the data to stable storage. This feature, referred to as “safe asynchronous writes,” can vastly reduce the number of disk I/O requests on the server, thus significantly improving write throughput.

The writes are considered "safe" because status information on the data is maintained, indicating whether it has been stored successfully. Therefore, if the server crashes before a commit operation, the client will know by looking at the status indication whether to resubmit a write request when the server comes back up.

4.5.2.2 Read throughput

NFS sequential read throughput, as measured at the client, is enhanced via the VMM read-ahead and caching mechanisms. Read-ahead allows file data to be transferred to the client from the NFS server in anticipation of that data being requested by an NFS client application. By the time the request for data is issued by the application, it is possible that the data resides already in the client's memory, and the request can be satisfied immediately. VMM caching allows re-reads of file data to occur instantaneously, assuming that the data was not paged out of client memory, which would necessitate retrieving the data again from the NFS server.

4.5.2.3 Tuning VMM for maximum caching of NFS data

NFS does not have a data-caching function, but the Virtual Memory Manager (VMM) caches pages of NFS data just as it caches pages of disk data. If a system is essentially a dedicated NFS server, it may be appropriate to permit the VMM to use as much memory as necessary for data caching. This is accomplished by setting the maxperm parameter, which controls the maximum percentage of memory occupied by file pages, to 100 percent. For example:

```
# vmtune -P 100
```

The same technique could be used on NFS clients, but would only be appropriate if the clients were running workloads that had very little need for working-segment pages.

4.5.2.4 The rsize and wsize options

The `mount` command provides some NFS tuning options that are often ignored or used incorrectly because of a lack of understanding of their use.

The most useful options are those for changing the read and write size values. These options define the maximum sizes of each RPC for read and write. Often, the rsize and wsize options of the `mount` command are decreased in order to decrease the read/write packet that is sent to the server. There can be two reasons why you might want to do this:

1. The server may not be capable of handling the data volume and speeds inherent in transferring the read/write packets (8 KB for NFS Version 2 and 32 KB for NFS Version 3). This might be the case if a NFS client is using a PC as an NFS server. The PC will likely have limited memory available for buffering large packets.
2. If a read/write size is decreased, there may be a subsequent reduction in the number of IP fragments generated by the call. If you are dealing with a faulty network, the chances of a call/reply pair completing with a two-packet exchange are greater than if there must be seven packets successfully exchanged. Likewise, if you are sending NFS packets across multiple networks with different performance characteristics, the packet fragments may not all arrive before the timeout value for IP fragments.

Reducing the rsize and wsize may improve the NFS performance in a congested network by sending shorter package trains for each NFS request. But a side effect is that more packets are needed to send data across the network, increasing total network traffic, as well as CPU utilization on both the server and client.

On the other hand, if your NFS file system is mounted across a high-speed network, such as the SP Switch, then larger read and write packet sizes would enhance NFS file system performance. With NFS Version 3, rsize and wsize can be set as high as 65536. The default is 32768. With NFS Version 2, the largest that rsize and wsize can be is 8192, which is also the default.

4.5.3 Tuning the numbers of nfsd and biod daemons

There is a single nfsd daemon and a single biod daemon, each of which is multithreaded (multiple kernel threads within the process). Also, the number of threads is self-tuning in that it creates additional threads as needed. You can, however, tune the maximum number of nfsd threads by using the nfs_max_threads parameter of the `nfso` command.

Determining the best numbers of nfsd and biod daemons is an iterative process. Guidelines can give you no more than a reasonable starting point. By default there are six biod daemons on a client and eight nfsd daemons on a server. The defaults are a good starting point for small systems, but should probably be increased for client systems with more than two users or servers with more than two clients. A few guidelines are as follows:

- In each client, estimate the maximum number of files that will be written simultaneously. Configure at least two biod daemons per file. If the files are large (more than 32 KB), you may want to start with four biod daemons per file to support read-ahead or write-behind activity. It is common for up to five biod daemons to be busy writing to a single large file.
- In each server, start by configuring as many nfsd daemons as the sum of the numbers of biod daemons that you have configured on the clients to handle files from that server. Add 20 percent to allow for non-read/write NFS requests.
- If you have fast client workstations connected to a slower server, you may have to constrain the rate at which the clients generate NFS requests.The best solution is to reduce the number of biod daemons on the clients, with due attention to the relative importance of each client's workload and response time.

After you have arrived at an initial number of biod and nfsd daemons, or have changed one or the other, do the following:

1. First, recheck the affected systems for CPU or I/O saturation with the `vmstat` and `iostat` commands. If the server is now saturated, you must reduce its load or increase its power, or both.
2. Use the command `netstat -s` to determine if any system is experiencing UDP socket buffer overflows. If so, and the system is not saturated, increase the number of biod or nfsd daemons.
3. Examine the nullrecv column in the `nfsstat -s` output. If the number starts to grow, it may mean there are too many nfsd daemons. However, this is less likely on this operating system's NFS servers than it is on other platforms. The reason for that is that all nfsd daemons are not awakened at the same time when an NFS request comes into the server. Instead, the first nfsd daemon wakes up, and if there is more work to do, this daemon wakes up the second nfsd daemon, and so on.

To change the numbers of nfsd and biod daemons, use the `chnfs` command.

To increase the number of biod daemons on the client may decrease server performance because it allows the client to send more requests at once, further overloading the network and the server. In extreme cases of a client overrunning the server, it may be necessary to reduce the client to one biod daemon, as follows:

```
# stopsrc -s biod
```

This leaves the client with the kernel process biod still running.

Note

When you configure more than 100 nfsd daemons on a uniprocessor system or more than 200 nfsd daemons on a SMP system, you will probably start to see NFS performance degradation, depending on the characteristics of the NFS traffic. The same rule can be applied to biod daemons.

4.5.4 Tuning to avoid retransmits

Related to the hard-versus-soft mount question is the question of the appropriate time-out duration for a given network configuration. If the server is heavily loaded, is separated from the client by one or more bridges or gateways, or is connected to the client by a WAN, the default time-out criterion may be unrealistic. If so, both server and client are burdened with unnecessary retransmits. For example, if the following command:

```
# nfsstat -c
```

```
Client rpc:
Connection oriented
calls      badcalls   badxids    timeouts   newcreds   badverfs   timers
36883      0          0          0          0          0          0
nomem      cantconn   interrupts
0          0          0
Connectionless
calls      badcalls   retrans    badxids    timeouts   newcreds   badverfs
20         0          0          0          0          0          0
timers     nomem      cantsend
0          0          0

Client nfs:
calls      badcalls   clgets     cltoomany
36880      0          0          0
Version 2: (0 calls)
null       getattr    setattr    root       lookup     readlink   read
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
wrcache    write      create     remove     rename     link       symlink
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
mkdir      rmdir      readdir    statfs
0 0%       0 0%       0 0%       0 0%
Version 3: (36881 calls)
null       getattr    setattr    lookup     access     readlink   read
0 0%       818 2%     0 0%       18 0%      1328 3%    0 0%       34533 93%
write      create     mkdir      symlink    mknod      remove     rmdir
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
rename     link       readdir    readdir+   fsstat     fsinfo     pathconf
0 0%       0 0%       0 0%       181 0%     2 0%       1 0%       0 0%
commit
0 0%
#
```

reports a significant number (greater than 5 percent of the total) of both time-outs and badxids, you could increase the timeo parameter with the following SMIT fast path:

```
# smitty chnfsmnt
```

Identify the directory you want to change, and enter a new value on the line NFS TIMEOUT (In tenths of a second). The default time is 0.7 seconds (timeo=7), but this value is manipulated in the NFS kernel extension depending on the type of call. For read calls, for example, the value is doubled to 1.4 seconds.

To achieve control over the timeo value for AIX V4 clients, you must set the nfs_dynamic_retrans option of the `nfso` command to 0. There are two directions in which you can change the timeo value, and in any given case, there is only one right way to change it. The correct way, making the time-outs longer or shorter, depends on why the packets are not arriving in the allotted time.

If the packet is only late and does finally arrive, then you may want to make the timeo variable longer to give the reply a chance to return before the request is retransmitted.

On the other hand, if the packet has been dropped and will never arrive at the client, then any time spent waiting for the reply is wasted time, and you want to make the timeo shorter.

One way to estimate which option to take is to look at a client's `nfsstat -cr` output and see if the client is reporting lots of badxid counts. A badxid value means that an RPC client received an RPC call reply that was for a different call than the one it was expecting. Generally, this means that the client received a duplicate reply for a previously retransmitted call. Packets are thus arriving late and the timeo should be lengthened.

Also, if you have a network analyzer available, you can apply it to determine which of the two situations is occurring. Lacking that, you can simply try setting the timeo option higher and lower and see what gives better overall performance. In some cases, there is no consistent behavior. Your best option then is to track down the actual cause of the packet delays/drops and fix the real problem (that is, a server or network/network device).

For LAN-to-LAN traffic through a bridge, try 50 (tenths of seconds). For WAN connections, try 200. Check the NFS statistics again after waiting at least one day. If the statistics still indicate excessive retransmits, increase timeo by 50 percent and try again. You will also want to examine the server workload and the loads on the intervening bridges and gateways to see if any element is being saturated by other traffic.

4.5.5 Dropped packets

Given that dropped packets are detected on an NFS client, the real challenge is to find out where they are being lost. Packets can be dropped at the client, the server, and somewhere on the network.

4.5.5.1 Packets dropped by the client

Packets are rarely dropped by a client. Because each client RPC call is self-pacing, that is, each call must get a reply before going on, there is little opportunity for overrunning system resources. The most stressful operation is probably reading, where there is a potential for 1 MB+/second of data flowing into the machine. While the data volume can be high, the actual number of simultaneous RPC calls is fairly small and each biod daemon has its own space allocated for the reply. Thus, it is very unusual for a client to drop packets. Packets are more commonly dropped either by the network or by the server.

4.5.5.2 Packets dropped by the server

Two situations exist where servers will drop packets under heavy loads:

- **Adapter driver**

  When an NFS server is responding to a very large number of requests, the server will sometimes overrun the interface driver output queue. You can observe this by looking at the statistics that are reported by the `netstat -i` command. Examine the columns marked Oerrs and look for any counts. Each Oerrs is a dropped packet. This is easily tuned by increasing the problem device driver's transmit queue size. The idea behind configurable queues is that you do not want to make the transmit queue too long, because of latencies incurred in processing the queue. But because NFS maintains the same port and XID for the call, a second call can be satisfied by the response to the first call's reply. Additionally, queue-handling latencies are far less than UDP retransmit latencies incurred by NFS if the packet is dropped.

- **Socket buffers**

  The second common place where a server will drop packets is the UDP socket buffer. Dropped packets here are counted by the UDP layer and the statistics can be seen by using the `netstat -p udp` command. Examine the statistics marked as UDP for the `socket buffer overflows` statistic.

NFS packets will usually be dropped at the socket buffer only when a server has a lot of NFS write traffic. The NFS server uses a UDP socket attached to NFS port 2049 and all incoming data is buffered on that UDP port. The default size of this buffer is 60,000 bytes. You can divide that number by the size of the default NFS write packet (8192) to find that it will take only eight simultaneous write packets to overflow that buffer. This overflow could occur with just two NFS clients (with the default configurations).

In this situation, there is either high volume or high burst traffic on the socket.

1. If there is high volume, a mixture of writes plus other possibly non-write NFS traffic, there may not be enough nfsd daemons to take the data off the socket fast enough to keep up with the volume. Recall that it takes a dedicated nfsd daemon to service each NFS call of any type.
2. In the high burst case, there may be enough nfsd daemons, but the speed at which packets arrive on the socket is such that they cannot wake up fast enough to keep it from overflowing.

Each of the two situations is handled differently.

1. In the case of high volume, it may be sufficient to just increase the number of nfsd daemons running on the system. Because there is no significant penalty for running with more nfsd daemons on a machine, try this solution first. In the case of high burst traffic, the only solution is to enlarge the socket, in the hope that some reasonable size will be sufficiently large enough to give the nfsd daemons time to catch up with the burst. Memory dedicated to this socket will not be available for any other use, so it must be noted that a tuning objective of total elimination of socket buffer overflows by making the socket larger may result in this memory being under utilized for the vast majority of the time. A cautious administrator will watch the socket buffer overflow statistic, correlate it with performance problems, and determine how large to make the socket buffer.
2. You might see cases where the server has been tuned and no dropped packets are arriving for either the socket buffer or the driver Oerrs, but clients are still experiencing time-outs and retransmits. Again, this is a two-case scenario. If the server is heavily loaded, it may be that the server is just overloaded and the backlog of work for nfsd daemons on the server is resulting in response times beyond the default time-out set on the client. See Section 4.5.9, “NFS tuning checklist” on page 79 for hints on how to determine if this is the problem. The other possibility, and the most likely problem if the server is known to be otherwise idle, is that packets are being dropped on the network.

4.5.5.3 Dropped packets on the network

If there are no socket buffer overflows or Oerrs on the server, and the client is getting lots of time-outs and retransmits and the server is known to be idle, then packets are most likely being dropped on the network. What do we mean here when we say the network? We mean a large variety of things, including media and network devices such as routers, bridges, concentrators, and the whole range of things that can implement a transport for packets between the client and server.

Anytime a server is not overloaded and is not dropping packets, but NFS performance is bad, assume that packets are being dropped on the network. Much effort can be expended proving this and finding exactly how the network is dropping the packets. The easiest way of determining the problem depends mostly on the physical proximity involved and resources available.

Sometimes the server and client are in close enough proximity to be direct-connected, bypassing the larger network segments that may be causing problems. Obviously, if this is done and the problem is resolved, then the machines themselves can be eliminated as the problem. More often, however, it is not possible to wire up a direct connection, and the problem must be tracked down in place.

4.5.6 Cache file system

The Cache File System (CacheFS) may be used to further enhance read throughput in environments with memory-limited clients, very large files, and/or slow network segments by adding the potential needed to satisfy read requests from file data residing in a local disk cache on the client.

When a file system is cached, the data read from the remote file system is stored in a cache on the local system, thereby avoiding the use of the network and NFS server when the same data is accessed for the second time. CacheFS is designed as a layered file system; this means that CacheFS provides the ability to cache one file system (the NFS file system, also called the *back-file* system) on another (your local file system, also called the *front-file* system). CacheFS works as follows:

1. After creating a CacheFS file system on a client system, the system administrator specifies which file systems are to be mounted in the cache.
2. When a user on the client attempts to access files that are part of the back file system, those files are placed in the cache. The cache does not get filled until a user requests access to a file or files. Therefore, the initial request to access a file will be at normal NFS speeds, but subsequent accesses to the same file will be at local JFS speeds.
3. To ensure that the cached directories and files are kept up to date, CacheFS periodically checks the consistency of files stored in the cache. It does so by comparing the current modification time to the previous modification time.
4. If the modification times are different, all data and attributes for the directory or file are purged from the cache, and new data and attributes are retrieved from the back-file system.

4.5.7 Reduced requests for file attributes

Because read data can sometimes reside in the cache for extended periods of time in anticipation of demand, clients must check to ensure their cached data remains valid if a change is made to the file by another application. Therefore, the NFS client periodically acquires the file's attributes, which includes the time the file was last modified. Using the modification time, a client can determine whether its cached data is still valid.

Keeping attribute requests to a minimum makes the client more efficient and minimizes server load, thus increasing scalability and performance. Therefore, NFS Version 3 was designed to return attributes for all operations. This increases the likelihood that the attributes in the cache are up to date, and thus reduces the number of separate attribute requests.

4.5.8 Disabling unused NFS ACL support

If your workload does not use the NFS access control list (ACL) support on a mounted file system, you can reduce the workload on both client and server to some extent by specifying the noacl option. This can be done as follows:

```
options=noacl
```

Set this option as part of the client's /etc/filesystems stanza for that file system.

4.5.9 NFS tuning checklist

Here is a checklist that you can follow when tuning NFS:

1. Check to see if you are overrunning the server.

   The general method is to see if slowing down the client will increase the performance. The following methods can be tried independently:

   a. Try running with just one biod daemon on the mount.

   b. Try running with just one biod daemon on the affected client. If performance increases, then something is being overrun either in the network or on the server. Run the `stopsrc -s biod` command to stop all the SRC biod daemons. It will leave one kernel process biod with which you can still run. See if it runs faster with just the one biod process. If it has no effect, restart the biod daemons with the `startsrc -s biod` command. If on the other hand, it runs faster, attempt to determine where the packets are being dropped when all daemons are running. Networks, network devices, slow server, overloaded server, or a poorly tuned server could all cause this problem.

   c. Try reducing the read/write sizes to see if things speed up.

   d. If you are using UDP, try using a TCP mount instead.

2. Check for Oerrs.

   This is probably the most common under-configuration error that affects NFS servers. If there are any Oerrs, increase the transmit queues for the network device. This can be done with the machine running, but the interface must be detached before it is changed (`rmdev -l`). You can not shut down the interface on a diskless machine, and you may not be at liberty to shut down the interface if other work is going on. In this case, you can use the `chdev` command to put the changes in ODM so they will be activated on the next boot. Start by doubling the current value for the queue length, and repeat the process (adding an additional 30 each time) until no Oerrs are reported.

On SP systems where NFS is configured to run across the high-speed switch, Oerrs may occur on the switch when there has been a switch fault and the switch was temporarily unavailable. An error is counted towards Oerr for each attempt to send on the switch that fails. These errors cannot be eliminated.

3. Look for any errors.

   Any counts that are very large indicate problems.

4. Check for NFS UDP/TCP buffer overruns.

   When using UDP, buffer overruns on the NFS server are another frequent under-configuration for NFS servers. TCP sockets can be similarly overrun on very busy machines. Tuning for both is similar, so this section discusses only UDP.

   Run the `netstat -s` command and examine the UDP statistics for the socket buffer overflows statistic. If it is anything other than 0, you are probably overrunning the NFS UDP buffer. Be aware, however, that this is the UDP socket buffer drop count for the entire machine, and it may or may not be NFS packets that are being dropped. You can confirm that the counts are due to NFS by correlating between packet drop counts on the client using the `nfsstat -cr` command to socket buffer overruns on the server while executing an NFS write test.

   Socket buffer overflows can happen on heavily stressed servers or on servers that are slow in relation to the client. Up to 10 socket buffer overflows are probably not a problem. Hundreds of overflows are. If this number continually goes up while you watch it, and NFS is having performance problems, NFS needs tuning.

   Two factors can tune NFS socket buffer overruns. First, try increasing the number of nfsd daemons that are being run on the server. If that does not solve the problem, you must adjust two kernel variables, sb_max (socket buffer max) and nfs_socketsize (the size of the NFS server socket buffer). Use the `no` command to increase sb_max. Use the `nfso` command to increase the nfs_socketsize variable.

   The sb_max parameter must be set larger than nfs_socketsize. It is hard to suggest new values. The best values are the smallest ones that also make the `netstat` report 0 or just produce a few socket buffer overruns.

Remember, in AIX V4, the socket size is set dynamically. Configurations using the `no` and `nfso` commands must be repeated every time the machine is booted. Add them in the /etc/rc.nfs file, right before the nfsd daemons are started and after the biod daemons are started. The position is crucial.

5. Check for mbuf problems.

   See if there are any requests for mbufs denied or delayed. If so, increase the number of mbufs available to the network.

6. Check for very small interpacket delays.

   There have been rare cases where this has caused problems with machines. If there is a router or other hardware between the server and client, you can check its documentation to see if the interpacket delays can be configured. If so, try increasing the delay.

7. Check for media mismatches.

   When packets are traversing two networks with widely different speeds, the router might drop packets when taking them off the high speed net and trying to get them out on the slower net. This has been seen particularly when a router was trying to take packets from a server on FDDI and send them to a client on Ethernet. It could not send out the packets fast enough on Ethernet to keep up with the FDDI. The only solution is to try to slow down the volume of client requests, use smaller read/write sizes, and limit the number of biod daemons that can be used on a single file system.

8. Check for MTU mismatches.

   Run the `netstat -i` command and check the MTU on the client and server. If they are different, try making them the same and see if the problem is eliminated. Also be aware that slow or wide area networks between the machines, routers, or bridges may further fragment the packets to traverse these network segments. Attempt to determine the smallest MTU between source and destination, and change the rsize/wsize on the NFS mount to some number lower than that lowest-common-denominator MTU.

9. Check for routing problems.

   Use the `traceroute` command to look for unexpected routing hops or delays.

10. Check for errlog entries.

    Run the `errpt` command and look for reports of network device or network media problems. Also look for any disk errors that might be affecting NFS server performance on exported file systems.

4.5.10 Examples of NFS configurations

The following are a couple of examples of how to determine the number of nfsd and biod daemons on SP nodes. If you have one server node for ten client nodes each mounting two file systems, you will need 120 nfsd daemons on the server. For example:

120 nfsd daemons = 10 X 6 X 2

If you have a client node mounting 5 NFS file systems from a NFS server, you will need 30 biod daemons on the client node. For example:

30 biod daemons = 5 X 6

If you have ten client nodes each mounting 5 NFS file systems all from the same server node, you may have a problem using the maximum number of biod daemons on each client node. Client nodes start at 30 biod daemons each server node needs 30 X 10 nfsd daemons or 300 nfsd daemons.

You may want to change the client nodes to each configure 20 biod daemons and change the server node to configure 200 nfsd daemons. This will work better, especially with a SMP NFS server node.

The other parameters that need addressing are the nfs_socketsize and nfs_tcp_socketsize parameters. These settings can be found using the `nfso` command. The current default values of 60000 are way too small for use on a SP system. These parameters specify the amount of space available in the NFS queue of pending requests. This queue is eventually read, and the requests handed off to a nfsd daemon. In the queue, write requests include the data to be written, so enough space to handle concurrent write requests to a server must be allocated. If this size is too small, then the client node will record NFS time-outs and retries as requests are dropped at the server.

Note

The current default value for nfs_socketsize and nfs_tcp_socketsize is 60000, but this value is too small for large SP system configuration. Start by doubling the value.

In NFS V2.0, you only have the nfs_socketsize. In NFS V3.0, since it now includes the ability of using TCP as the transport mechanism between client and server, you must make sure that both these parameters are set to the correct size. To determine the appropriate size on the server, you need to calculate the maximum number of clients that will be writing to the server at the same time, times the number of file systems they will be writing to, times 6, which is the maximum number of biod's per remote file system. By taking this number, and multiplying it by 4K (NFS V2.0) or 32K (NFS V3.0), you can estimate the queue space needed.

If the size needed for the nfs_socketsize and nfs_tcp_socketsize are very large, you might want to reduce the number of biod daemons per mounted file system from the client nodes. If you do, then you can reduce the size determined above by using the smaller number of biod daemons per file system, rather than the value of 6.

The following are a couple of examples of how to set the socket sizes for NFS:

If you have ten client nodes running NFS V2.0, each writing to two file systems on a single server node, you need to set nfs_socketsize to 491520 or 480 KB. For example:

10 X 6 X 2 X 4096 = 480 KB

If you have ten client nodes, each running NFS V3.0 over TCP writing to two file systems on a single server node, you need to set nfs_socketsize and nfs_tcp_socketsize to 39321600 or 3.8 MB. For example:

10 X 6 X 2 X 32768 = 3.8 MB.

There is a possible problem with sb_max size in `no` tunables. If sb_max is less than the socket size you set, the sb_max value is really the limit for the NFS socket.

You might want to change the client node mount parameters to use two biod daemons on each mount to the server node. This will result in a nfs_tcp_socketsize of 1310720 or 1.28 MB. For example:

10 X 2 X 2 X 32768 = 1.28 MB.

For more detailed information regarding tuning NFS, refer to the *AIX V4.3 System Management Guide: Communications and Networks*, SC23-4127.

4.6 SP system-specific tuning recommendations

The SP system usually requires that tunable settings be changed from the default values in order to achieve optimal performance of the entire system. How to determine what these settings are is described in the sections that follow. However, where to set these tunable values is very important. If they are not set in the correct places, subsequent rebooting of the nodes or other changes can cause them to change, or be lost.

For all dynamically tunable values (those that take effect immediately), the setting for each node should be set in the tuning.cust file. This file is found in the /tftpboot directory on each node. There is also a copy of the file in the same directory on the CWS. Tunables changed using the `no`, `nfso` or `vmtune` commands can be included in this file. Even though the sample files do not include `nfso` and `vmtune` commands, they can be added here with no problems.

There are a small number of tuning recommendations that are not dynamically tunable values that need to be changed in the /etc/rc.net file. These tunables are for ARP cache tuning and setting the number of adapter types per interface. The following tunables are the only ones that should be added to rc.net:

- arptab_nb
- arptab_bsize
- ifsize

There are several reasons why the tuning.cust file should be used rather than rc.net for dynamically tunable settings:

- If you damage /etc/rc.net, you can render the node unusable, requiring a reinstallation of the node.
- If you partially damage /etc/rc.net, getting to the node through the console connection from the CWS can take several minutes or even over an hour. This is because part of the initialization of the node is to access remote nodes or the CWS, and because /etc/rc.net is defective, each attempt to get remote data takes nine minutes to time out and fail (time-out defined in the system).
- If you damage /tftpboot/tuning.cust file, at least the console connection will work, enabling you to log in through the CWS and fix the bad tunable settings.
- If you decide to create your own inittab entry to call a file with the tuning settings, future releases of PSSP will require a tuning.cust set of tunables to be run, overriding your local modifications.
- Tuning.cust is run from /etc/rc.sp, so it will always be run on a reboot.
- Tuning.cust includes a stop and start of inetd as of PSSP Release 2.4, which is required for all daemons to inherit the SP-specific tuning settings.

Using the sample tuning.cust settings selected as part of the install, the SP nodes will at least function well enough to get up and running for the environment type selected.

If the system has nodes that require different tuning settings, it is recommended that a copy of each setting be saved on the CWS. When nodes with specific tuning settings are installed, that version of tuning.cust needs to be moved into /tftpboot on the CWS.

Another option is to create one tuning.cust file that determines the node number, and based on that node number, sets the appropriate tuning values.

4.6.1 IBM-supplied files of SP tunables

When a node is installed, migrated or customized, and that node's boot/install server does not have a /tftpboot/tuning.cust file, a default file of performance tuning variable settings /usr/lpp/ssp/install/tuning.default is copied to /tftpboot/tuning.cust on that node. You can choose from one of the IBM-supplied tuning files, or you can create or customize your own. There are four sample tuning files currently available. The existing files are located in the /usr/lpp/ssp/install/config directory, and are as follows:

- *tuning.commercial* contains initial performance tuning parameters for a typical commercial environment.
- *tuning.development* contains initial performance tuning parameters for a typical interactive and/or development environment. These are the default tuning parameters.
- *tuning.scientific* contains initial performance tuning parameters for a typical engineering/scientific environment.
- *tuning.server* contains initial performance tuning parameters for a typical server environment.

The other option is to create and select your own alternate tuning file. While this may not be the initial choice, it certainly must be the choice at some point in time. On the CWS, create a tuning.cust file, or you can begin with an IBM-supplied file. Edit the tuning.cust file with your favorite editor, making sure changes are saved. Once you are finished, proceed to the installation of nodes. This tuning.cust file is then propagated to each node's /tftpboot/tuning.cust file from the boot/install server when the node is installed, migrated, or customized and is maintained across reboots.

4.7 Tuning the SP network for specific workloads

This section describes four typical environments: Software Development, Scientific and Technical, Commercial Database, and Server Configuration. The settings given are only initial settings and are not guaranteed to be optimized for your environment. They will get the system up and running fairly well. You should look at your specific implementation and adjust your tuning settings accordingly.

4.7.1 Tuning for development environments

The typical development environment on the SP consists of many users all developing an application or running small tests of a system. On the SP system, this type of environment has lots of connections between the nodes using TCP/IP. In this case, setting the TCP/IP parameters to more conservative values is an approach to prevent exhaustion of the switch buffer pool areas. Most of the high-speed tuning is not done in this environment because single connection performance is not critical. What is important is that aggregate requirements from several developers do not exhaust system resources.

Note

In a typical development environment, only small packets are used, but lots of sockets are active at one time.

Table 8 provides network tunable settings designed as initial values for a development environment. These settings are only initial suggestions. Start with them and understand you may need to change them. Remember to keep track of your changes and document them.

Table 8. Software development tuning parameters

Parameter	Value
thewall	16384 (higher value in AIX 4.3.2 and later. See Section 4.3, “AIX network tunables” on page 45 for more details.)
sb_max	131072
subnetsarelocal	1
ipforwarding	0 (set 1 only if needed)
tcp_sendspace	65536
tcp_recvspace	65536

Parameter	Value
udp_sendspace	32768
udp_recvspace	65536
rfc1323	1
tcp_mssdflt	1448
tcp_pmtu_discover	0
udp_pmtu_discover	0
ipqmaxlen	512

The way these initial values are derived is that the network traffic expected consists of small packets with lots of socket connections. The tcp_sendspace and tcp_recvspace parameters are kept small so that a single socket connection cannot use up lots of network buffer space, causing buffer space starvation. It is also set so that high performance for an individual socket connection is not expected. However, if lots of sockets are active at any one time, the overall resources will enable high aggregate throughput over the switch.

4.7.2 Tuning for scientific and technical environments

The typical scientific and technical environment usually has only a few network sockets active at any one time, but sends large amounts of data. The following information sets up the network tunables so that a single socket connection or a few connections can get the full SP Switch bandwidth. In doing this, however, you can cause problems on small packet networks like Ethernet and Token Ring. This is the trade-off that has to be made to get peak performance out of the SP system.

The following are the characteristics for a scientific and technical environment:

- Few active connections
- Large TCP windows size (tcp_sendspace, tcp_recvspace)
- rfc1323 turned on
- Switch buffer pools larger than TCP window (`chgcss`)
- Large TCP socket write size (socket write call)

To achieve peak data transfer across the SP switch using TCP/IP you need to increase the tunables that affect buffer sizes, queue sizes and the TCP/IP window. To get the best TCP/IP transfer rate, you need to size the TCP/IP window large enough to keep data streaming across a switch without stopping the IP stream. The switch has an MTU of 65520 bytes. This is the largest buffer of data that it can send. When using TCP/IP, TCP will send as many buffers as it can until the total data sent without acknowledgment from the receiver reaches the tcp_sendspace value.

Experimentation has found that having at least six times the size of the MTU of the SP switch (or SP switch2) available in the TCP window size allows TCP/IP to reach high transfer rates. However, if you set tcp_sendspace and tcp_recvspace to 655360 bytes, it can hurt the performance of the other network adapters connected to the node. This can cause adapter queue overflows, as described in Section 5.1, “Adapter queue size” on page 95.

The settings in Table 9 are only initial suggestions. Start with them and understand you may need to change them. Remember to keep track of your changes and document them.

Table 9. Scientific and technical environment tuning parameters

Parameter	**Value**
thewall	16384 (higher value in AIX 4.3.2 and later. See Section 4.3, “AIX network tunables” on page 45 for more details.)
sb_max	1310720
subnetsarelocal	1
ipforwarding	0 (set 1 only if needed)
tcp_sendspace	655360
tcp_recvspace	655360
upd_sendspace	65536
upd_recvspace	655360
rfc1323	1
tcp_mssdflt	Varies (depending on other network types)
tcp_pmtu_discover	0
udp_pmtu_discover	0
ipqmaxlen	512

Now we list the most common problems that occur in the scientific and technical environment. Most of them are explained in more detail later in this book.

- TCP/IP pacing caused by incorrect TCP window tuning (the Nagle Algorithm). See Section 4.2.9, “Description of the Nagle Algorithm” on page 42.
- Single server / multiple client scaling problems. See Section 9.6, “Single-server multiple-client node problems” on page 197.
- Synchronization point file server accesses.
- File systems performance, including NFS (see Section 4.5.1, “NFS overview for performance” on page 69) and GPFS (Chapter 14, “GPFS” on page 369).
- ARP cache size on greater than 128 node systems. See Section 4.2.7.1, “ARP cache tuning” on page 40.
- Turning on tcp_pmtu_discover can increase route table and increase lookup time on large systems.

4.7.3 Tuning for commercial and database environments

The typical commercial and database environments generally have many network connections between nodes. For environments with lots of active connections, the tcp_sendspace and tcp_recvspace need to be adjusted so that the aggregate amount of the TCP window across all connections does not exceed the available buffer space for the SP Switch. In commercial and database environments where only a few connections are active, you can increase the tcp_sendspace and tcp_recvspace sizes to get better per-connection performance over the switch.

The following are the characteristic for commercial and database environments:

- Lots of active connections.
- Smaller TCP or dynamically adjusted window sizes (tcp_sendspace, tcp_recvspace).
- Aggregate active TCP window sizes less than switch pools.
- TCP write sizes aligned along switch pool buffer sizes (socket write call).

These settings in Table 10 are only initial suggestions. Start with them and understand you may need to change them. Remember to keep track of your changes and document them.

Table 10. Commercial and database environment tuning parameters

Parameter	Value
thewall	16384 (higher value in AIX 4.3.2 and later. See Section 4.3, “AIX network tunables” on page 45 for more details.)
sb_max	1310720
subnetsarelocal	1
ipforwarding	0 (set 1 only if needed)
tcp_sendspace	262144
tcp_recvspace	262144
udp_sendspace	65536
udp_recvspace	655360
rfc1323	1
tcp_mssdflt	1448
tcp_pmtu_discover	0
udp_pmtu_discover	0
ipqmaxlen	512

The way these initial values are derived is that the network traffic expected consists of small packets with lots of socket connections. However, when running a parallel database product, you want to be able to get as much SP Switch throughput to a single connection as you can without causing problems on other network adapters. The settings in Table 10 are also designed to enable a single socket to be able to send to an Ethernet adapter without causing adapter queue overruns. In addition, the tcp_sendspace and tcp_recvspace are large enough to get a majority of the switch bandwidth at database size packets.

If other applications, with vastly different network characteristics, are run on the same node, such as TSM (Tivoli Storage Manager), or data mining type applications, that tend to use few sockets, these settings may not provide peak performance. In these cases, the TCP window settings may have to be increased. Conflicts with the settings needed by TSM can be resolved by having TSM do its own socket level tuning. See Chapter 12, “Tivoli Storage Manager (ADSM)” on page 337 for more information.

The following list describes the most common problems that occur in the commercial and database environment. Most of them are explained in more detail later in this book.

- TCP/IP pacing caused by incorrect TCP window tuning (the Nagle algorithm). See Section 4.2.9, “Description of the Nagle Algorithm” on page 42.
- Single server / multiple client scaling problems. See Section 9.6, “Single-server multiple-client node problems” on page 197.
- Switch pool sizing. See Section 5.1, “Adapter queue size” on page 95.
- Excessive aggregate memory requirements.
- Adapter queue overruns. See Section 9.2, “Adapter queue overflows” on page 192.
- Poor applications IP traffic characteristics.
- Unbalanced I/O channel traffic.
- Scheduled operations tuning requirements conflicts with ongoing database workload.
- Turning on tcp_pmtu_discover can increase route table and increase lookup time on large systems.

4.7.4 Tuning for server environments

The server environment usually is a node serving a lot of data to one or many other nodes on an SP system. It can also be serving data to machines outside the SP system through gateway nodes. This environment puts the highest demands on getting the aggregate amount of traffic for the SP Switch or TCP/IP buffer pools. If a server node in an SP system is potentially serving hundreds of requests or connections, tcp_sendspace and tcp_recvspace need to be small. This prevents a large number of large data requests from consuming the entire switch and TCP/IP buffer pools.

In systems where there is one server and the rest of the nodes run an application that needs larger tcp_sendspace and tcp_recvspace sizes, it is acceptable to use different settings on the appropriate nodes. In this situation, the nodes talking to each other use large TCP windows for peak performance, and when talking to the server, use small windows. The effective TCP window is the least common denominator of the tcp_sendspace and tcp_recvspace values.

The following are the characteristics for tuning the server environments:

- Large buffer pools for staging remote traffic.
- Large number of daemons to handle remote requests.
- Multiple disks and adapters to distribute I/O.
- Smaller TCP windows to prevent exhaustion of buffer pools (tcp_sendspace and tcp_recvspace).

Table 11 provides tunable network settings designed as initial values for server environments.

Table 11. Server tuning parameters

Parameter	Value
thewall	65536 (See Section 4.3, “AIX network tunables” on page 45 for more details.)
sb_max	1310720
subnetsarelocal	1
ipforwarding	0 (set 1 only if needed)
tcp_sendspace	65536
tcp_recvspace	65536
udp_sendspace	65536
udp_recvspace	655360
rfc1323	1
tcp_mssdflt	1448
tcp_pmtu_discover	0
udp_pmtu_discover	0
ipqmaxlen	512

The way these initial settings are derived is that the network traffic expected is coming from many connections. To prevent exhaustion of the TCP/IP buffer pools or the switch pools, the tcp_sendspace, the tcp_recvspace and the sb_max parameters are reduced. If the total number of active requests from the server are small, these values may be increased to allow more buffer area per connection. This increase will help improve performance for small numbers of connections as long as aggregate TCP window space does not exceed other buffer areas.

We describe the most common problems that occur in the server environment. Most of them are explained in more details later in this book.

- Exhaustion of switch pools due to use of large TCP windows. See Section 5.1, “Adapter queue size” on page 95 for more details.
- Insufficient buddy buffers for VSD and GPFS. See Chapter 13, “VSD” on page 353 and Chapter 14, “GPFS” on page 369 for more details.
- Adapter queue overflows on non-switch adapters. See Section 5.1, “Adapter queue size” on page 95 for more details.
- Not enough NFS daemons or too small NFS socket size. See Section 9.4, “NFS performance in SP system” on page 194 for more details.
- Bottlenecks on a single I/O bus or subset of disks.
- Too small memory size.

4.7.5 Summary of workload tunables

Table 12 gives a combined overview of our tunables for the different environments.

Table 12. Summary of workload tunables

Parameter	Dvlpmnt.	S&T	Commercial	Server
thewall	16384	16384	16384	65536
sb_max	131072	1310720	1310720	1310720
subnetsarelocal	1	1	1	1
ipforwarding	0	0	0	0
tcp_sendspace	65536	655360	262144	65536
tcp_recvspace	65536	655360	262144	65536
udp_sendspace	32768	65536	65536	65536
udp_recvspace	65536	655360	655360	655360
rfc1323	1	1	1	1
tcp_mssdflt	1448	Varies (depending on other network types)	1448	1448
tcp_pmtu_discover	0	0	0	0

Parameter	Dvlpmnt.	S&T	Commercial	Server
udp_pmtu_discover	0	0	0	0
ipqmaxlen	512	512	512	512

Chapter 5. Adapter tuning

Many types of network adapters are supported in an RS/6000 SP environment. When data is sent, it is passed through the TCP/IP layers to the device drivers. Therefore, tuning the network adapter is critical to maintain peak throughput for network traffic.

5.1 Adapter queue size

The high throughput of a switch can cause problems with network adapters, such as Ethernet, Token Ring and FDDI, connected to nodes acting as gateways on SP systems. There is a fixed number of adapter queue slots to stage packets in each network adapter device driver for traffic to that network. The transmit adapter queue length specifies the maximum number of packets for the adapter. The SP Switch send and receive pools are separate buffer pools as shown in Figure 13.

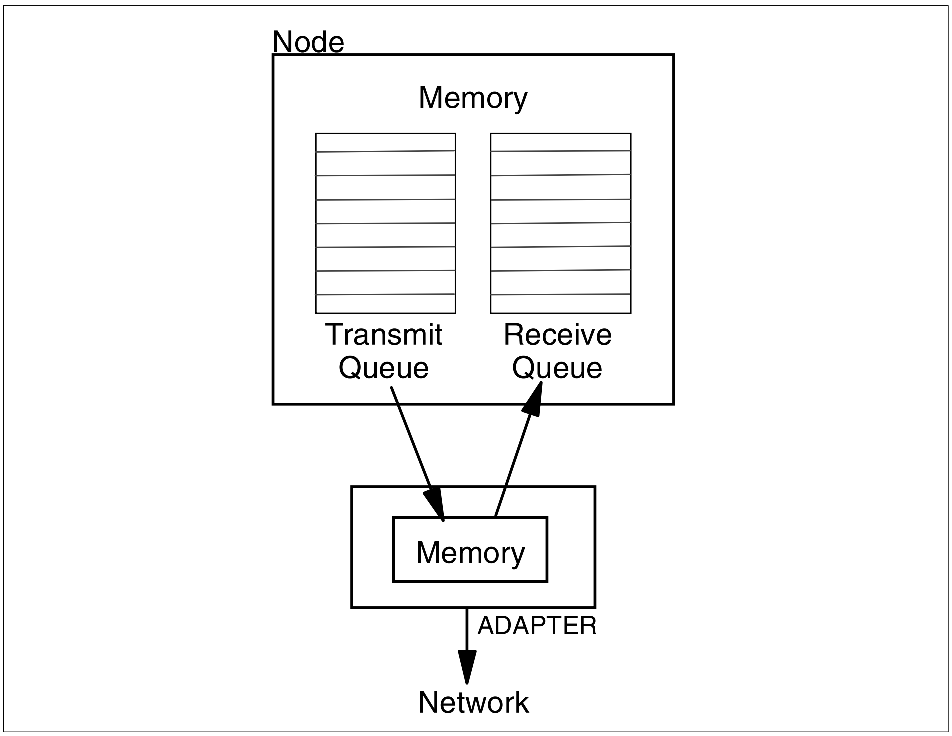

Figure 13. Adapter queue overview

If the adapter queue size is exceeded, subsequent packets are discarded by the adapter device driver, resulting in dropped packets. This results in a transmit time-out in the TCP layer, which leads to a rollback of the TCP window and the resending of data. For UDP, the result is lost packets.

Adapter queue overflows can be detected by looking at the errors logged in the adapter counters as "S/W Transmit Queue Overflows." For Ethernet, Token Ring, FDDI and ATM, the adapter statistics can be seen by using the `entstat`, `tokstat`, `fddistat` and `atmstat` commands.

Most communication drivers provide a set of tunable parameters to control transmit and receive resources. These parameters typically control the transmit queue and receive queue limits, but may also control the number and size of buffers or other resources. They limit the number of buffers or packets that may be queued for transmit or limit the number of receive buffers that are available for receiving packets. For an example, see Table 13; these parameters (for AIX 4.2.1 and later) can be tuned to ensure enough queueing at the adapter level to handle the peak loads generated by the system or the network.

Table 13. Transmit queue size examples

Adapter		Default	Range
MCA	Ethernet	512	20 - 2048
	10/100 Ethernet	64	16,32,64,128,256
	Token Ring	99 or 512	32 - 2048
	FDDI	512	3 - 2048
	ATM / 155 ATM	512	0 - 2048
PCI	Ethernet	64	16,32,64,128,256
	10/100 Ethernet	256 - 512	16,32,64,128,256
	Gigabit Ethernet	512	512-2048
	Token Ring	96 - 512	32 - 2048
	FDDI	30	3 - 250
	155 ATM	100	0 - 4096

5.1.1 Transmit and receive queues

For transmit, the device drivers may provide a *transmit queue* limit. There may be both hardware queue and software queue limits, depending on the driver and adapter. Some drivers have only a hardware queue, some have both hardware and software queues. Some drivers control the hardware queue internally and only allow the software queue limits to be modified. Generally, the device driver will queue a transmit packet directly to the adapter hardware queue. If the system CPU is fast relative to the speed of the network, or on an SMP system, the system may produce transmit packets faster than they can be transmitted on the network. This will cause the hardware queue to fill. Once the hardware queue is full, some drivers provide a software queue and subsequent packages will be queued to it. If the software transmit queue limit is reached, then the transmit packets are discarded. This can affect performance because the upper level protocols must then retransmit the discarded packets.

A typical example would be that you set your adapter queue length to 30. Assuming that the MTU of that adapter is 1500, you have set the maximum amount of data that the adapter can hold to 45,000 bytes. This is less than a single packet from a switch. Figure 14 illustrates the different MTU ratios. If you try and stage more packets to an adapter, then the packets that arrive when this queue is full get thrown away.

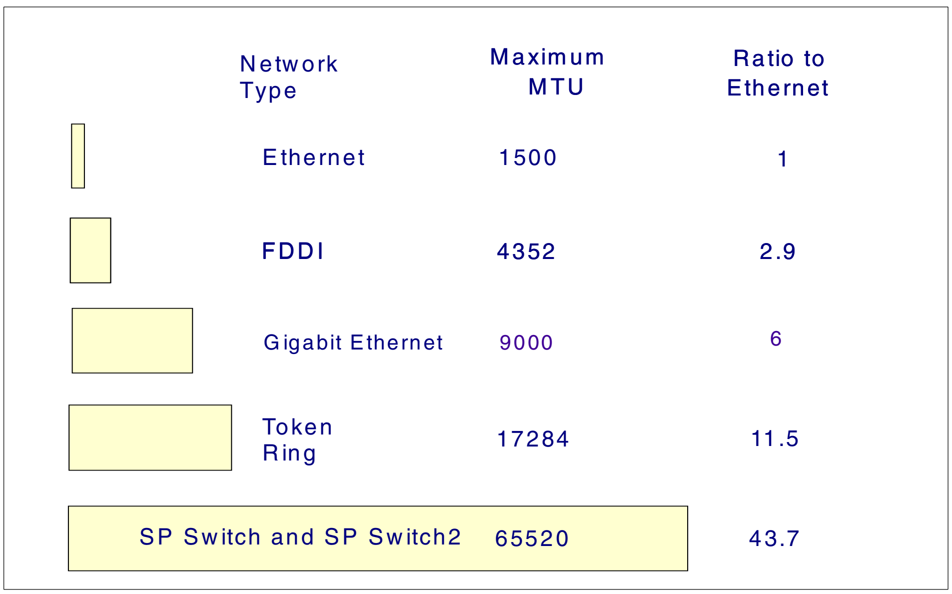

Figure 14. MTU ratio

Receive queues are the same as transmit hardware queues.

5.1.2 Displaying adapter queue settings

To show the adapter configuration settings, you can use the `lsattr` command or SMIT. For example, to display the default values of the settings, you can use the command:

```
# lsattr -D -l <adapter - name>
```

and to display the current values, you can use:

```
# lsattr -E -l <adapter - name>
```

Finally, to display the range of legal values of an attribute (for example, xmt_que_size) for a given adapter (for example, Token Ring), you can use the command:

```
# lsattr -R -l tok0 -a xmt_que_size
```

Different adapters have different names for these variables. For example, they may be named sw_txq_size, tx_que_size, or xmt_que_size, to name a few for the transmit queue parameter. The receive queue size and/or receive buffer pool parameters may be named rec_que_size, rx_que_size*,* or rv_buf4k_min*,* and so on.

Figure 15 shows the output of the `lsattr -E -l atm0` command on an IBM PCI 155 Mbps ATM adapter. This shows sw_xq_size set to 250 and the rv_buf4K_min receive buffers set to 48.

```
ma-mem         0x400000   N/A                                       False
regmem         Oxlff88000 Bus Memory address of Adapter Registers   False
virtmem        Oxlff90000 Bus Memory address of Adapter Virtual Mem False
busintr        3          Bus Interrupt Level                       False
intr_priority  3          Interrupt Priority                        False
use_alt_addr   no         Enable ALTERNATE ATM MAC address          True
alt_addr       OX0        ALTERNATE ATM MAC address (12 hex digits) True
sw_txq_size    250        Software Transmit Queue size              True
max_vc         1024       Maximum Number of VCs Needed              True
min_vc         32         Minimum Guaranteed VCs Supported          True
rv_buf4k_min   0x30       Minimum 4K-byte premapped receive buffer  True
interface_typ  0          Sonet or SH interface                     True
adapter_clock  1          Provide SONET Clock                       True
uni_vers       autot_detect N/A                                     True
```

Figure 15. lsattr command output for an ATM adapter

5.1.3 Changing adapter settings

The easiest way to change the adapter settings is by using SMIT. The other method is to use the `chdev` command.

For example, to change tx_que_size on en0 to 512, use the following sequence of commands. Note that this driver only supports four different sizes, so it is better to use SMIT to see the valid values.

```
# ifconfig en0 detach

# chdev -l ent0 -a tx_que_size=512

# ifconfig en0 up
```

5.1.4 Adapter tuning recommendations

If you consistently see output errors when running the `netstat -i` command, increasing the size of the xmt_que_size parameter may help. Check also the adapter transmit average overflow count. As a rule of thumb, always set xmt_que_size to the maximum.

One way to tune IP to prevent exceeding the adapter queue size is to reduce the aggregate TCP window size or udp_sendspace so that it is less than the transmit queue size times the segment size (MTU) on the network. This usually results in optimal throughput and slightly higher system overhead for network traffic. If multiple connections are sharing an adapter at the same time, the aggregate TCP window size across all connections should be slightly less than the transmit queue size times the segment size for the network media type.

5.2 SP Switch and SP Switch2 adapter tuning

The switch network is something unique for an SP system. It provides high performance connections for all nodes. Therefore, the right tuning of the switch adapters is essential for good overall system performance.

5.2.1 Switch adapter pools

Since PSSP Level 2.1 PTF 11, two variables are provided to tune the sizes of the switch adapter pools:

- rpoolsize - Size of SP Switch device driver receive pool in bytes.
- spoolsize - Size of SP Switch device driver send pool in bytes.

Note

To apply the changes, you need to reboot the node(s). Use the `chgcss` command to change the settings.

These pools are used to stage the data portion of IP packets for the switch. However, the allocation and sizes utilized from the pools can cause buffer starvation problems. The send pool and receive pool are separate buffer pools, one for outgoing data (send pool) and one for incoming data (receive pool). When an IP packet is passed to the switch interface, then if the size of the data is large enough, a buffer is allocated from the pool. If the amount of data fits in the IP header mbuf used in the mbuf pool, no send pool space is allocated for the packet. The amount of data that will fit in the header mbuf is a little less than 200 bytes, depending on what type of IP packet is being sent. The header size varies between UDP and TCP, or having rfc1323 turned on.

To see the current send pool and receive pool buffer sizes, as shown in Figure 16, issue the `lsattr -E -l css0` command.

```
# lsattr -E -l css0
bus_mem_addr   0x04000000 Bus memory address        False
int_level      0xb        Bus interrupt level       False
int_priority   3          Interrupt priority        False
dma_lvl        8          DMA arbitration level     False
spoolsize      2097152    Size of IP send buffer    True
rpoolsize      2097152    Size of IP receive buffer True
adapter_status css_ready  Configuration status      False
```

Figure 16. Viewing send and receive pool buffer sizes

In this particular example, the pool sizes have been increased from the default size of 512 KB to 2 MB. The pool settings can be changed using the `chgcss` command, and require rebooting the node.

For example, to change the size of both pools to 1 MB, enter:

```
# chgcss -l css0 -a rpoolsize=1048576 -a spoolsize=1048576
```

The SP switch adapter pools reside in pinned kernel memory. The SP Switch2 adapter has a 16 MB memory block physically installed, referred as rambus RDRAM, that enables software to copy user data directly to the adapter. With this on-board memory, the SP Switch2 adapter eliminates the need to access main storage and decreases the number of memory bus transfers by a factor of 2X to 3X.

5.2.2 Window sizes

With PSSP 3.2, we now have three new ODM attributes to specify bounds on window sizes and memory usage:

- win_poolsize - Total pinned node memory available for all user-space receive FIFOs.
- win_maxsize - Maximum memory used per window.
- win_minsize - Minimum memory used per window.

Note

No reboot is necessary to apply the changes.

All applications on a RS/6000 SP can use the switch (SP Switch or SP Switch2) network. Using the switch network, you can profit from a low latency and high bandwidth communication method. Two communication protocols are offered on the switch adapter:

- Standard TCP/IP communication through AIX sockets or message passing libraries.
- Dedicated user-space access via message passing libraries.

A distinct communication port between an application on the SP node and the switch is called a *window*. Each window has its own send FIFO and receive FIFO and a set of variables that describes the status of the FIFOs. The latter is used to properly transfer data to and from the window's FIFOs and the adapter.

There are different types of windows, as shown in Table 14.

Table 14. Window types

IP	This reserved window is responsible for the IP communication among nodes.
Service	This reserved window manages configuration and monitoring of the switch network.
VSD	This window will be reserved for VSD communication, if you install VSD on your system and you use LAPI or KLAPI.
User Space	These windows permit high-speed data communication among user applications.

Note

SP Switch2 has 16 user-space windows available. For SP Switch, however, there are 4 user-space windows available. The total number of available windows is 17 for SP Switch2 and 5 for SP Switch.

The total device memory reserved for all switch adapters UserSpace windows used as interface network FIFO buffers is specified by the new win_poolsize ODM attribute. The two other new attributes, win_maxsize and win_minsize represent maximum and minimum memory used per window. Note that these three new attributes are dynamically changeable, that is, no reboot is needed. In addition, these attributes pertain to both SP Switch and SP Switch2.

Recall that in PSSP earlier than level 3.2, only spoolsize and rpoolsize attributes could be changed using the `chgcss` command. Now this command is enhanced to include the three new attributes mentioned above. For example, to change the maximum window size to 1 megabyte, enter either of the commands shown in Figure 17:

```
# chgcss -l css0 -a win_maxsize=0x100000

or

# chgcss -l css0 -a win_maxsize=1048576
```

Figure 17. chgcss command

See Table 15 for the ODM default settings for the new attributes.

Table 15. settings for new attributes

Attribute	**Default value (SPS2/SPS)** [a]	**Min. value**	**Max. value (SPS2/SPS)** [b]
win_poolsize	128MB/80MB	maximum_windows * win_minsize	256MB/80MB
win_maxsize	16MB/16MB	256KB	win_poolsize/16MB
win_minsize	1MB/1MB	256KB	(win_poolsize/maximum_windows)/(win_poolsize/maximum_windows)

a. SPS2 is for SP Switch2 and SPS is for SP Switch.
b. SPS2 is for SP Switch2 and SPS is for SP Switch.

Take a look at Figure 18 for `lsattr -El css0` command output that verifies the current settings of the switch adapter available windows.

```
adapter_memory 0xe0000000         adapter memory address            False
rambus_memory  0x10c0000000       RAMBUS memory address             False
win_poolsize   134217728          Total window memory pool size     True
win_maxsize    16777216           Maximum window memory size        True
win_minsize    1048576            Minimum window memory size        True
int_priority   3                  Interrupt priority                False
int_level      32                 Bus interrupt level               False
spoolsize      2097152            Size of IP send buffer            True
rpoolsize      2097152            Size of IP receive buffer         True
khal_spoolsize 524288             Size of KHAL send buffer          True
khal_rpoolsize 524288             Size of KHAL receive buffer       True
adapter_status css_ready          Configuration status              False
diags_prog     /usr/sbin/diag     Diagnostic program                True
ucode_version  3                  Micro code version                True
ucode_name     /etc/microcode/col_ucodeMicro code name              True
window         VSD AVAIL AVAIL AVAIL AVAIL window owners            True
driver_debug   0                  Device Driver Debug               True
ip_chksum      off                if_cl checksum                    True
ip_debug       0                  if_cl debug level                 True
proto_debug    off                HAL debug                         True
```

Figure 18. lsattr -El css0 output

This is the output that you get with an SP Switch2 Adapter installed in your system.

5.2.3 Switch window reservation

There is also a way to reserve user-space windows for specific clients. For example, you want to reserve a window for GPFS or VSD. What you need to type is described in the following:

Syntax:

- `chgcss -l css0 -a window=token1{/token2{/token3{/token4}}}`

where:

- `token1=CMD:{RESERVE|RELEASE|QUERY}`
- `token2=ID:<client_name>`
- `token3=TYPE:{user_client|kernel_client}`
- `token4=COUNT:<window_count>`

Note

Always use the `chgcss` command to apply the changes. There is no reboot necessary.

5.2.3.1 How to reserve a window

Here is an example on how to reserve a window for GPFS:

```
# chgcss -l css0 -a window=cmd:reserve/id:GPFS/type:user_client/count:1
1
# chgcss -l css0 -a window=cmd:QUERY/id:GPFS
1
```

In this example, window 1 has been reserved for GPFS.

5.2.3.2 How to query all windows

To query the list of reserving applications for all windows, enter:

```
# chgcss -l css0 -a window=cmd:query
VSD GPFS AVAIL AVAIL AVAIL
```

In this example, window 0 is reserved for VSD, window 1 is reserved for GPFS, and windows 2, 3 and 4 are unreserved. With SP Switch2, you will see the following output:

```
VSD GPFS AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL
AVAIL AVAIL AVAIL AVAIL AVAIL
```

This occurs because of the higher amount of available windows with the SP Switch2 (totalling 16). The maximum number of windows available would be 17, if VSD or GPFS were not installed.

5.2.3.3 How to query for unreserved windows

To query the list of unreserved applications for all windows, enter:

```
# chgcss -l css0 -a window=cmd:QUERY/id:AVAIL
2 3 4
or enter:
# chgcss -l css0 -a window=cmd:QUERY/id:
2 3 4
```

In this example, windows 2, 3 and 4 are unreserved.

5.2.3.4 How to release windows

You can also release the reserved windows, enter the following:

```
# chgcss -l css0 -a window=cmd:RELEASE/id:GPFS
VSD AVAIL AVAIL AVAIL AVAIL
```

In this case, we released the reserved GPFS window and now only window 0 is still reserved for VSD.

5.2.4 Switch pool allocation

The size of the buffers allocated by the switch device driver starts at 4096 bytes, and increases to 65536 bytes in values of powers of 2. See Figure 19 for an overview.

If the size of the data being sent is just slightly larger than one of these sizes, the buffer allocated from the pool is the next size up. This can cause as low as 50 percent efficiency in usage of the buffer pools. More than half of the pool can go unused in bad circumstances.

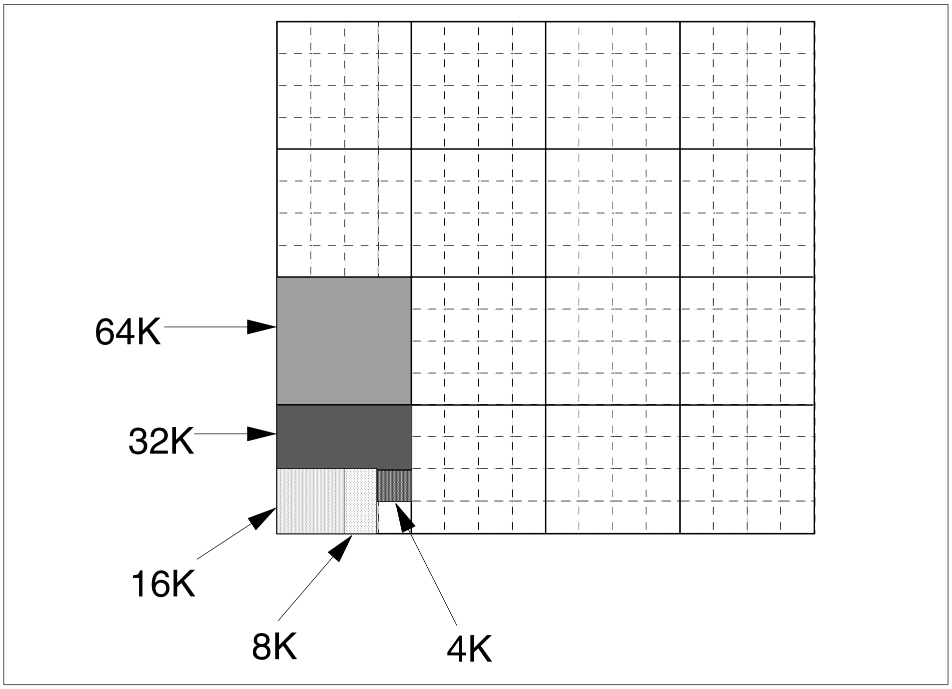

Figure 19. Switch pool allocation

When assembling TCP/IP packets, there is always one mbuf from the IP mbuf pool used to assemble the packet header information in addition to any data buffers from the spool. If the mbuf pool size is too small, and the system runs out of mbufs, the packet is dropped. The mbuf pool is used globally for all IP traffic, and set using the thewall tunable with the `no` command.

When sending 4 KB of data over the switch, an mbuf from the mbuf pool will be used, as well as one 4 KB spool buffer for the data. If the amount of data being sent is less than 200 bytes, no buffer from the spool is allocated, because there is space in the mbuf used for assembling the headers to stage the data. However, if sending 256 bytes of data, you will end up using one mbuf for the IP headers, and one 4 KB send pool buffer for the data. This is the worst case scenario, where you are wasting 15/16 of the buffer space in the send pool. These same scenarios apply to the receive pool when a packet is received on a node.

The key for peak efficiency of the spool and rpool buffers is to send messages that are at or just below the buffer allocation sizes, or less than 200 bytes.

5.2.5 Switch buffer pool allocation considerations

When tuning the rpool and spool, it is important to know the expected network traffic. As we have seen, if the size of the buffers for the applications is not ideal, much of the spool and rpool will be wasted. This can cause the need for a larger rpool and spool because of inefficient usage. When allocating the rpool and spool, realize that this space is pinned kernel space in physical memory. This takes space away from user applications and is particularly important in small memory nodes.

If there are a small number of active sockets, then there is usually enough rpool and spool space that can be allocated. In systems where a node has a large number of sockets opened across the switch, it is very easy to run out of spool space when all sockets transmit at once. For example, 300 sockets, each sending 33 KB of data, will far exceed the 16 MB limit for the spool. Or, 1100 sockets, each sending 1 KB packets, will also exceed the maximum limit. But this is different with the new SP Switch2. The new switch adapter allows us to allocate 32 MB of rpool and spool size. The SP Switch2 adapter will not pin the memory, as it was done before with the SP Switch adapter. We now have 32 MB physically installed in the new adapter. That makes it much faster.

Note

The new SP Switch2 receive pool and send pool maximum size is 32 MB instead of 16 MB.

On the receive side of a parallel or client/server implementation, where one node acts as a collector for several other nodes, the rpool runs into the same problem. Four nodes, each with 600 sockets, each sending to one node two 1 KB packets, will exceed the rpool limit, but those same sockets, each sending twice as much data, 4 KB in one 4 KB packet, will work. The key here is sending a single larger packet rather than several smaller ones.

Another situation that aggravates exhaustion of the pools is SMP nodes. Only one CPU is used to manage the send or receive data streams over the switch. However, each of the other CPUs in the SMP node is capable of generating switch traffic. As the number of CPUs increases, so does the aggregate volume of TCP/IP traffic that can be generated. For SMP nodes, the spool size should be scaled to the number of processors when compared to a single CPU setting.

5.2.6 Sizing send and receive pool requirements

When trying to determine the appropriate rpool and spool sizes, you need to get a profile of the message sizes that are being sent by all applications on a node. This will help to determine how the rpool and spool will be allocated in the total number of buffers. At a given pool size, you will get 16 times as many buffers allocated out of the pool for 4 KB messages as for 64 KB messages.

Once you have a profile of the packet or buffer sizes used by all applications using the switch on a node, you can then determine roughly how many of each size spool or rpool buffers will be needed. This then determines your initial rpool and spool settings.

The sizes allocated from the pool are not fixed. At any point in time, the device driver will divide the pool up into the sizes needed for the switch traffic. If there is free space in the send pool, and smaller buffers than the current allocation has available are needed, then the device driver will carve out the small buffer needed for the current packet. As soon as the buffer is released, it is joined back with the 64 KB buffer it came from. The buffer pool manager tries to return to 64 KB block allocations as often as possible to maintain high bandwidth at all times. If the pool were fragmented, and a large buffer needed 64 KB, then there may not be 64 KB of contiguous space available in the pool. Such circumstances would degrade performance for the large packets. If all buffer space is used, then the current packet is dropped, and TCP/IP or the application needs to resend it, expecting that some of the buffer space was freed up in the meantime. This is the same way that the transmit queues are managed for Ethernet, Token Ring and FDDI adapters. If these adapters are sent more packets than their queues can handle, the adapter drops the packets.

The upper limit for the SP Switch send pool and receive pool is 16 MB each. This means you can get a maximum of 4096 4 KB or 256 64 KB buffers each for sending and receiving data.

The new SP Switch2 adapter offers a 32 MB limit for both the send pool and receive pool. This gives us a maximum of 8192 4 KB or 512 64 KB buffers each for sending and receiving data.

Use the `vdidl3xx` commands shown in Table 16 to check for IP pool size problems indicated by slow network traffic or ENOBUF errors.

Table 16. vdidl3xx commands

SP Switch adapter type	Command
TB3	`vdidl3 -i`
SPSMX	`vdidl3mx -i`
TB3PCI	`vdidl3pci -i`

Note

There is no `vdidl3xx` command available right now for the SP Switch2.

For possible receive pool problems, check the `errpt` output and look for "mbuf pool threshold exceeded" entries for css0 device.

Figure 20 on page 108 is a sample output of the first table from the `vdidl3` command.

```
#/usr/lpp/ssp/css/vdidl3 -i
send pool: size=524288 anchor@=0x50002000 start@=0x50e70000
tags@=0x50c1c200
bkt   allocd     free  success     fail    split     comb    freed
 12        0        0      409        0      316        0        0
 13        0        0      220        0      161        0        0
 14        0        0        0        0        0        0        0
 15        0        0        0        0        0        0        0
 16        0        8        0        0        0        0        0
```

Figure 20. Output of the vdidl3 command

The fields of the `vdidl3` command are as follows:

bkt Lists the pool allocation in power of 2 for the line it is on. The line starting with 12 means 2 to the 12th or 4 KB allocations. The line starting with 16 means 2 to the 16th or 64 KB allocations.

allocd Lists the current allocations at the time the command was run, for each of the size allocations in the first column. This is an instantaneous value and, therefore, can increase and decrease from one execution of the command to the next.

free Lists the number of buffers of each allocation that are allocated and unused at the time the command was run. In the above example, there are eight 64 KB allocations free for use. This is an instantaneous value and, therefore, can increase and decrease from one execution of the command to the next.

success Increments every time an allocation of the given size succeeded. This counter is cumulative and, therefore, shows the number of successes since the adapter was last initialized.

fail Increments every time an allocation is not available for the size requested.This counter is cumulative and, therefore, shows the number of fails since the adapter was last initialized.

split Indicates how many times the allocation size was extracted from the pool by carving the size needed from a larger allocation in the pool. This counter is cumulative and, therefore, shows the number of splits since the adapter was last initialized.

comb Currently not used.

freed Currently not used.

5.2.7 Sample switch send pool size estimate

In this section, we calculate the switch pool requirements for a common distribution of packet sizes. In our example, the node stages a mix of packet sizes, of which 25 percent are smaller than 200 bytes, 50 percent are 5 KB, and 25 percent are 32 KB.

Buffer size profile:

Less than 200 bytes	25 percent
5 KB packets	50 percent
32 KB packets	25 percent

Aggregate tcp_sendspace equivalent in packets: 1024 packets

Total send pool space:

No space needed for small packets	0 MB
512 - 8 KB buffers for 5 KB packets	4 MB
256 - 32 KB buffers for 32 KB packets	8 MB

____

Total send pool space needed 12 MB

If the number of packets staged at any one time is 1024, then the spool initial setting should be 12582912 or 12 MB. None of the small packets needs any spool space (if the amount of data fits in the IP header mbuf used in the mbuf pool, no send pool space is allocated for the packet); the 5 KB packets each use 8 KB out of the spool and need about 4 MB of space, and the 32 KB packets need about 8 MB of spool space. The total estimated pool size needed is 12 MB.

The above calculation is a conservative estimate in that it assumes all packets will be staged at once. In reality, as packets are being staged into the pool, other packets are being drained out, so the effective number of active buffers should be less.

5.2.8 Reducing send/receive pool requirements

Reducing the TCP window size across sockets reduces the amount of send pool buffer space required on the sender side, and the receive pool buffer space required on the receiver side. Realize, however, that reducing the window size could affect the switch performance by limiting the maximum amount of outstanding data. Consider this as a trade-off. The exact setup will depend on the application requirements.

Also, reducing the number of sockets sending data through the switch simultaneously will reduce send pool requirements. This is true when the same amount of data must be sent using fewer sockets; this will not reduce the requirement and probably will not affect the overall performance of the application.

Finally, be aware that changes on the sender side will affect the receiver, so always keep in mind that tuning involves both sides of a network connection.

5.3 SP Ethernet tuning

There are three areas to consider when tuning an SP Ethernet network:

- Number of frames
- Number of nodes
- Number of networks

The lower the amount of traffic in each Ethernet network, the better the performance or response time of the system. If you run a small system of fewer than two frames, you probably connected the SP network to your own network through a gateway. For larger configurations, we suggest that you dedicate one node per frame as a gateway to the rest of the SP system. Routing from the gateway node can be either through the SP Ethernet or across a switch.

In the case where an Ethernet switch is used, the best way to connect external Ethernets to the SP is through separate Ethernet adapters on nodes that route the external traffic over the switch. Having too many external Ethernet connections route through the SP Ethernet can lead to congestion. This can cause the system to slow down.

When routing across a switch, you should give careful consideration to other traffic moving across the switch. An environment which has many parallel jobs will suffer in performance if a lot of the user traffic is routed through the SP Switch adapter to that same node.

In larger SP system configurations, the network topology and the name server can cause problems. If there are many name server queries at the same time, and the topology from a node to the name server is complex, performance can suffer. Under these circumstances, there are three possible solutions:

1. Create a name server within the SP system complex so you are not at risk for traffic problems to an external name server.
2. Do not use the name server, but create an /etc/hosts file of all addresses that the SP system and the applications need, and change the nodes to resolve their addresses locally.
3. Specify the name lookup hierarchy to first look locally in the /etc/hosts file, then the name server. Use the /etc/netsvc.conf file to specify that. The file should contain a line like `hosts=local,bind`. Since AIX 4.1 you can also use the environment variable NSORDER (`NSORDER=local,bind`).

5.4 Gigabit ethernet performance tuning recommendations

Instead of just having a maximum MTU size of 1500 bytes, such as the standard Ethernet Adapters (10/100 Mbit Ethernet and similar), this Gigabit Ethernet offers a 9000 byte MTU size.

- The 9000 byte MTU size (called Jumbo Frame) gives you a TCP throughput that can be more than twice as high as the standard 1500 byte MTU size.

5.5 Token-Ring performance tuning recommendations

The default MTU of 1492 bytes is appropriate for token rings that interconnect to Ethernets or to heterogeneous networks in which the minimum MTU is not known.

- Unless the LAN has extensive traffic to outside networks, the MTU should be increased to 8500 bytes. This allows NFS 8 KB packets to fit in one MTU. Further increasing the MTU to the maximum of 17000 bytes seldom results in corresponding throughput improvement.
- The application block size should be in multiples of 4096 bytes.

5.6 FDDI performance tuning recommendations

Despite the comparatively small MTU, this high-speed medium benefits from substantial increases in socket buffer size.

- Unless the LAN has extensive traffic to outside networks, the default MTU of 4352 bytes should be retained.
- Where possible, an application using TCP should write multiples of 4096 bytes at a time (preferably 8 KB or 16 KB) for maximum throughput.

5.7 ATM performance tuning recommendations

Unless the LAN has extensive traffic to outside networks, the default MTU of 9180 bytes should be retained. ATM traffic routed over the SP Switch will benefit from MTUs up to 64 KB.

- Where possible, an application using TCP should write multiples of 4096 bytes at a time (preferably 8 KB or 16 KB) for maximum throughput.

5.8 HIPPI performance tuning recommendations

The default MTU of 65536 bytes should not be changed.

- Where possible, an application using TCP should write 65536 bytes at a time for maximum throughput.
- Set sb_max to a value greater than 2*655360.
- TCP and UDP socket send and receive space defaults should be set to more than 64 KB.

5.9 Escon interface tuning

To achieve peak TCP/IP throughput over a switch and through Escon to MVS, you need to make sure that the maximum packet size possible is used over the Escon connection. Table 17 lists all necessary parameters to tune for maximum packet size of 4096 bytes.

Table 17. Escon interface tuning parameters

Parameter	MVS/ TCP/IP	AIX Escon gateway node	Explanation
Segment Size	4096	_	Maximum packet size.
DATABUFFERSIZE	256K	_	The DATABUFFERSIZE variable on MVS TCP/IP must be set to 256 KB.
Escon Interface MTU	_	4096	On the Escon gateway node, set the Escon interface MTU to 4096.
ipforwarding	_	1	On the Escon gateway node, set ipforwarding to 1.
tcp_mssdflt	_	4056	On the nodes across the switch from the Escon gateway node, set tcp_mssdflt to 4056.

These settings ensure that the maximum packet size across the interface will be a full Escon interface packet of 4096 bytes. These settings generally enable peak throughput over the Escon interface. Recent releases of TCP/IP on MVS support window scaling, known as rfc1323. You may be able to increase the window size of the connection by setting rfc1323 to 1 on the SP. You want to avoid setting the TCP window larger than 256 KB, which is the recommended maximum buffer area on the MVS side of the socket connection.

Chapter 6. Tuning other components in AIX

Since performance tuning of an SP system is primarily about tuning the network resources in AIX, other system components, such as memory, CPU and I/O, that are not specific for SP systems, will be discussed separately.

To be able to performance tune the components covered in this chapter, you need experience with how different workloads and applications in production environments utilize specific system designs and implementations based on RS/6000 computers. Not only will different production systems behave and perform differently, the same system will usually change its resource utilization during the course of a production cycle.

Each section in this chapter will first discuss the basics of each resource or resource manager and then continue with how to performance tune each component.

The sections are:

- Section 6.1, “Memory” on page 115
- Section 6.2, “CPU” on page 129
- Section 6.3, “I/O” on page 141

Note

Manipulating resource utilization settings discussed in this chapter can severely affect performance and availability of the system if done inappropriately and should be done with caution in a production environment.

If possible, use a separate test system to evaluate proper resource settings before performing changes in the production environment.

6.1 Memory

Memory is a critical resource in an SP system. Insufficient memory, or poor use of memory, usually results in performance degradation and, in some cases, even serious availability problems.

Note

When calculating memory requirements for parallelized applications that will run on an SP system, be careful how to estimate the memory consumption for each node.

A working collective of SP nodes with a parallel application can be compared conceptually to the equivalent running on an SMP system, when it comes to calculating the memory requirements.

For example a parallel RDBMS that would need 4 GB memory on a SMP system would not necessarily need 4 GB on each SP node, if four nodes were used and database tables and client loads were distributed amongst these four nodes (usually 1 GB would be needed per node [for the RDBMS]). The result is 4 GB for the SP environment, but 1 GB for each node.

For additional info on the specific tuning commands, see Chapter 10, “Resource monitoring” on page 203, Chapter 11, “Tools” on page 243, and the *AIX Command Reference*, GBOF-1802, or the *Understanding IBM RS/6000 Performance and Sizing*, SG24-4810.

6.1.1 Virtual memory manager

The Virtual Memory Manager (VMM) provides the virtual memory facilities that are used by the other parts of the system to implement the following:

- Virtual address space of processes
- Sharing of executables
- Shared memory segments
- Mapped files

The VMM implements virtual memory, by dividing it into segments. The segments are then divided into fixed-size units called pages. Each page in a segment can be in physical memory or stored on disk until it is needed. When a process accesses a page that is not present in physical memory, the VMM reads the page into memory; this is called a *page in*. When physical memory is not available, the VMM writes pages to disk; this is called a *page out* or *page steal*.

The following are some of the segment types:

Working storage Segments that are used to implement the data areas of processes and shared memory segments. The pages for working storage segments are *stored in the paging spaces* configured in the system.

Persistent storage Segments that are used to manipulate files and directories. When a persistent storage segment is accessed, the pages are *read and written from its file system*.

Client storage Segments that are used to implement some virtual file systems like Network File System (NFS) and the CD-ROM file system. The storage for client segment pages can be in a *local or remote computer*.

6.1.2 Real-memory management

Virtual-memory segments are partitioned into fixed-size units called *pages*. The page size is 4096 bytes. Each page in a segment can be in real memory, or stored on disk until it is needed. Similarly, real memory is divided into 4096-byte *page frames*. The role of the VMM is to manage the allocation of real-memory page frames and to resolve references by the program to virtual-memory pages that are not currently in real memory or do not yet exist. Because the amount of virtual memory that is in use at any given instant can be larger than real memory, the VMM must store the surplus on disk. From the performance standpoint, the VMM has two, somewhat opposed, objectives:

- Minimize the overall processor time and disk bandwidth cost of the use of virtual memory.
- Minimize the response time cost of page faults.

In pursuit of these objectives, the VMM maintains a free list of page frames that are available to satisfy a page fault. The VMM uses a page replacement algorithm to determine which virtual memory pages currently in memory will have their page frames reassigned to the free list.

6.1.2.1 Persistent versus working segments

Since each page of a persistent segment has a permanent disk storage location, the VMM writes the page back to that location when the page has been changed and can no longer be kept in real memory. If the page has not changed, its frame is simply reassigned to the free list. If the page is referenced again later, a new copy is read in from its permanent disk storage location.

Working segments are transitory, exist only during their use by a process, and have no permanent disk storage location. Process stack and data regions are mapped to working segments, shared library text and data segments. Pages of working segments must have disk-storage locations to occupy when they cannot be kept in real memory. The disk paging space is used for this purpose.

Persistent-segment types are further classified. Client segments are used to map remote files (for example, files that are being accessed through NFS/GPFS), including remote executable programs. Pages from client segments are saved and restored over the network to their permanent file location, not on the local disk paging space. Journaled and deferred segments are persistent segments that must be automatically updated. If a page from a journaled or deferred segment is selected to be removed from real memory (paged out), it must be written to disk paging space unless it is in a state that allows it to be committed (written to its permanent file location).

Table 18 outlines the three types of real memory and their backing stores.

Table 18. Backing store for different memory segment types

Real memory	**Backing store**
Persistent	File space
Working	Paging space
Client	Network to file space

6.1.2.2 Computational versus file memory

Computational memory, also known as computational pages, consists of the pages that belong to working-storage segments or program text segments. A segment is considered to be a program text segment if an instruction cache miss occurs on any of its pages.

File memory (or file pages) consists of the remaining pages. These are usually pages from permanent data files in persistent storage.

6.1.2.3 Page replacement

When the number of available real memory frames on the free list becomes low, the page stealer is invoked. The page stealer moves through the Page Frame Table (PFT), looking for pages to steal.

Page replacement is done directly within the scope of the thread if running on an uniprocessor. On an multiprocessor system, page replacement is done through the lrud kernel process, which is dispatched to a CPU when the minfree threshold has been reached.

Starting with AIX 4.3.3, the lrud kernel process is multithreaded with one thread per memory pool. Real memory is split into evenly sized memory pools based on the number of CPUs and the amount of RAM. The number of memory pools will be as follows:

MAX (Number of CPUs/8, RAM in GB/16)

But the number of memory pools will not be more than the number of CPUs and not less than 1.

6.1.2.4 minfree and maxfree

The number of page frames on the free list is controlled by the following parameters:

minfree	Minimum acceptable number of real-memory page frames in the free list. When the size of the free list falls below this number, the VMM begins stealing pages. It continues stealing pages until the size of the free list reaches maxfree.
maxfree	Maximum size to which the free list will grow by VMM page-stealing. The size of the free list may exceed this number as a result of processes terminating and freeing their working-segment pages or the deletion of files that have pages in memory.

The VMM attempts to keep the size of the free list greater than or equal to minfree. When page faults or system demands cause the free list size to fall below minfree, the page replacement algorithm runs. The size of the free list must be kept above a certain level (the default value of minfree) for several reasons. For example, the operating system's sequential-prefetch algorithm requires several frames at a time for each process that is doing sequential reads. Also, the VMM must avoid deadlocks within the operating system itself, which could occur if there were not enough space to read in a page that was required to free a page frame.

Choosing minfree and maxfree settings

As mentioned before, the purpose of the free list is to keep track of real memory page frames released by terminating processes and to supply page frames to requestors immediately, without forcing them to wait for page steals and the accompanying I/O to complete. The minfree limit specifies the free-list size, below which page stealing to replenish the free list is to be started. The maxfree parameter is the size above which stealing will end.

The objectives in tuning these limits are to ensure that:

- Any activity that has critical response-time objectives can always get the page frames it needs from the free list.
- The system does not experience unnecessarily high levels of I/O because of premature stealing of pages to expand the free list.

The default value of minfree and maxfree depend on the memory size of the machine. The default value of maxfree is determined by this formula:

maxfree = minimum (# of memory pages/128, 128)

By default, the minfree value is the value of maxfree - 8. However, the difference between minfree and maxfree should always be equal to or greater than maxpgahead. Or in other words, the value of maxfree should always be greater than or equal to minfree plus the size of maxpgahead.

The minfree/maxfree values will be different if there is more than one memory pool; the minfree/maxfree values shown by the `vmtune` command will be the sum of the minfree/maxfree for all memory pools.

Remember, that minfree pages in some sense are wasted, because they are available, but not in use. If you have a short list of the programs you want to run fast, you can investigate their memory requirements with the `svmon` command, and set minfree to the size of the largest. This technique risks being too conservative, because not all of the pages that a process uses are acquired in one burst. At the same time, you might be missing dynamic demands that come from programs not on your list that may lower the average size of the free list when your critical programs run.

If you observe the page frame consumption of your applications, either during initialization or during normal processing, you will soon have an idea of the number of page frames that need to be in the free list to keep the program from waiting for memory.

How to set minfree and maxfree

The following example sets minfree to 128 and maxfree to 144 pages respectively:

```
# /usr/samples/kernel/vmtune -f 128 -F 144
```

Since the purpose of increasing minfree and maxfree are to ensure that processes needing free pages will get them without having to wait for a page reclamation cycle, it is important to understand how much memory on average are needed by applications in the current workload.

One simple basic setting for minfree and maxfree is to calculate a starting point as follows:

- minfree = (120 * # of CPUs)
- maxfree to ((maxpageahead * # of CPUs) + minfree)

As an example we use a node with 8 CPUs, which would give us a minfree of 960 and a maxfree of 1024 (with the default value for maxpageahead).

The difference between minfree and maxfree usually needs to be larger than the maxpageahead value. However, care must be taken, since the more memory that needs to be freed each time, the longer the wait each time, but if chosen carefully, the throughput might increase.

Gung ho reclamation

Since page reclamation does not occur immediately when a process is terminated, it can force it to happen by increasing the minfree and maxfree values. Be aware, though, that this should be done in increments. If you increase it too much and your system is heavily loaded, you will kick-start the page reclamation and it will not stop until it has freed minfree and maxfree number of pages.

Depending on how heavily loaded your system is, this can lead to several minutes of searching, freeing and paging, and since this is a prioritized process, it will effectively make your multiuser AIX system into a single user system for the duration of this process. This in turn means that all user applications will end up in the run queue, and some might even get errors from system calls and time-outs on communication links. If the applications (maybe server processes) can not handle these error situations, they will abort, and their services will become unavailable in much the same way as when a low paging space situation results.

In the following example, we have a system with 4 GB memory and we will increment minfree and maxfree until we see heavy page reclamation every 30 seconds. The test should be running with three windows active: one for entering the `vmtune` commands, one for monitoring using `vmstat`, and the third with a `vmtune` command ready to reset the values to the starting point. If you happen to go too far, the response time can extend up to a couple of minutes, and if you need to, you just have to hit Enter in the window with the resetting `vmtune` command, and wait for the full page reclamation cycle to settle down again.

But before starting, you should monitor the system for a while using vmstat and understand how many pages it is searching and freeing every second. If it is constantly searching many times more than it is freeing, and it is freeing a couple of hundred/thousands (depending on the number of CPUs in the node), you should be very cautious when trying this out.

```
# cd /usr/samples/kernel
# integer maxpageahead minfree
# vmtune|awk 'NR==4{print $4,$5}'|read maxpageahead minfree
# while :; do
> set -x
> vmtune -f $minfree -F $(($minfree+$maxpageahead))
> set +x
> read
> ((minfree=$minfree*2))
> done
```

In the preceding example, you should run it as a script and, the first time, verify that is working by printing the vmtune line instead of executing it (just insert print as the first command on the line). Let the system settle with the current values before incrementing them higher (one to ten minutes should be sufficient). If you change them too quick, you might not have seen the full effect yet.

6.1.2.5 Repaging

A page fault is considered to be either a new page fault or a repage fault. A new page fault occurs when there is no record of the page having been referenced recently. A repage fault occurs when a page that is known to have been referenced recently is referenced again and is not found in memory because the page has been replaced (and perhaps written to disk) since it was last accessed.

A perfect page-replacement policy would eliminate repage faults entirely (assuming adequate real memory) by always stealing frames from pages that are not going to be referenced again. Thus, the number of repage faults is an inverse measure of the effectiveness of the page-replacement algorithm in keeping frequently reused pages in memory, thereby reducing overall I/O demand and potentially improving system performance.

To classify a page fault as new or repage, the VMM maintains a repage history buffer that contains the page IDs of the N most recent page faults, where N is the number of frames that memory can hold. For example, 512 MB memory requires a 128 KB repage history buffer. At page-in, if the page's ID is found in the repage history buffer, it is counted as a repage. Also, the VMM estimates the computational-memory repaging rate and the file-memory repaging rate separately by maintaining counts of repage faults for each type of memory. The repaging rates are multiplied by 0.9 each time the page-replacement algorithm runs, so that they reflect recent repaging activity more strongly than historical repaging activity.

6.1.2.6 minperm and maxperm

The following thresholds are expressed as percentages. They represent the fraction of the total real memory of the machine that is occupied by file pages (pages of noncomputational segments).

minperm — If the percentage of real memory occupied by file pages falls below this level, the page-replacement algorithm steals both file and computational pages, regardless of repage rates.

maxperm — If the percentage of real memory occupied by file pages rises above this level, the page-replacement algorithm steals only file pages.

When the percentage of real memory occupied by file pages is between minperm and maxperm, the VMM normally steals only file pages, but if the repaging rate for file pages is higher than the repaging rate for computational pages, computational pages are stolen as well. The main intent of the page-replacement algorithm is to ensure that computational pages are given fair treatment. For example, the sequential reading of a long data file into memory should not cause the loss of program text pages that are likely to be used again soon. The page-replacement algorithm's use of the thresholds and repaging rates ensures that both types of pages get treated fairly, with a slight bias in favor of computational pages.

strict_maxperm — strict_maxperm is implemented so that when the number of file pages exceeds maxperm, VMM will not give out any more free frames to persistent segments (threads will be waited) and page-replacement will run to free up only file pages. However, if the number of file pages is within a threshold of maxperm (the threshold is minfree) but has not yet reached it, a free frame will be given for a persistent segment and page-replacement will be started for file pages. While page-replacement is running in this threshold, it will free anything unless the number of file pages is greater than maxperm.

Choosing minperm and maxperm settings

As mentioned before, the purpose of minperm and maxperm is to ensure that enough memory is available to satisfy both the demand for memory by starting new processes and for I/O read ahead and write-behind caching (collectively named file caching in this section). It can be set to favor file caching or having as many programs in memory simultaneously.

Note

maxperm is not a strict limit; it is only considered when the VMM needs to perform page replacement, unless strict_maxperm is activated.

With the default parameters, the system would try to maintain up to ((RAMSIZE-4MB)*.8), or approximately 80 percent of real memory with persistent pages, leaving only ((RAMSIZE-4MB)*.2), approximately 20 percent, for working pages.

When memory becomes full of file pages, the VMM is spending a lot of time looking for dirty pages to write back to disk during `sync` processing or when the application is using O_SYNC[1] or fsync() logic. In these cases, VMM scans all pages in real memory associated with the file to determine which pages must be written to disk. Since the file has many pages in memory, CPU cycles are wasted looking at each page in memory.

Before deciding on changing the settings of these parameters, the current workload and memory usage must be known. For some workloads, it is more important to use as much as possible for file caching, and for some workloads, with less I/O but more computational properties, to ensure that program text is favored. To find out how much memory is currently used for file mapping, run `vmtune` and look at numperm:

[1] With the open() subroutine

```
# /usr/samples/kernel/vmtune
vmtune:  current values:
  -p       -P        -r          -R         -f       -F       -N        -W
minperm  maxperm  minpgahead maxpgahead  minfree  maxfree  pd_npages maxrandwrt
 209508   838032       2          8        120      128     524288        0

  -M      -w      -k      -c        -b         -B          -u        -l    -d
maxpin npswarn npskill numclust numfsbufs hd_pbuf_cnt lvm_bufcnt lrubucket defps
838852   73728   18432       1      93          80          9      131072     1

        -s              -n         -S           -h
sync_release_ilock  nokilluid  v_pinshm  strict_maxperm
        0               0          0             0

number of valid memory pages = 1048565  maxperm=79.9% of real memory
maximum pinable=80.0% of real memory    minperm=20.0% of real memory
number of file memory pages = 14127     numperm=1.3% of real memory
```

To determine how many page frames that are actually used in the current production system, use the `svmon -G` command (for more details of this command, see Chapter 10, “Resource monitoring” on page 203 and Chapter 11, “Tools” on page 243):

```
# svmon -G

               size       inuse        free         pin     virtual
memory      1048565       93469      952627     1048565       37759
pg space    2359296         423

               work        pers        clnt
pin           56734         156           0
in use        79147       14322           0
```

In the `vmtune` output, you can compare the number of file memory pages with pers in use from `svmon`. In these examples, they differ, since they were not run at the same time.

How to set minperm and maxperm

First get a current snapshot with `vmtune` and look at the current values for maxperm, minperm and numperm. If numperm is in between maxperm and minperm then file pages or computational pages will be kicked out to memory when the minfree value is reached and until maxfree is reached. However, if numperm is above maxperm, then file pages will get kicked out. We usually need to set up maxperm so that file pages get kicked out so that we can keep computational pages longer, since those are the ones going out to disk and being paged in and out (therefore causing thrashing). numperm is a value that changes with time and is not tunable via `vmtune`. It will change with I/O load on the system.

If you would like to keep more recently read files in memory, such as on a database server, you would raise minperm towards maxperm. numperm will then approach minperm as I/O is applied to the system and not go below it.

As default, a file cache is configured with about 20 percent (minperm) and 80 percent (maxperm). This is too high for operation with a file system-based database, since the data files are not only available in the file system, but are also in the data buffer. This results in an unnecessarily large amount of paging, which affects the performance. The following are a few guidelines[2]:

- For file system-based database servers with up to 1.5 GB memory, set minperm to 5 and maxperm to 10.
- For file system-based database servers with more than 1.5 GB memory, set minperm to 3 and maxperm to 8.
- For dedicated NFS servers, use the default or increase maxperm to 100.
- For other use, set maxperm to 60 and then decrease maxperm by increments of 5 (not less than 10) while monitoring numperm with `vmtune`.

However, in some cases, when an application creates large files, on an otherwise busy system and I/O problems occur, maxperm is set too low and needs to be increased (at least during this type of job).

Note

With the default option for strict_maxperm of 0, maxperm is only used when page replacement needs to occur (when minfree threshold is reached).

When set to 1, the strict_maxperm option will case the maxperm parameter to be a hard limit on the amount of RAM that can be used as a persistent file cache. This can be very useful in the cases where there is double-buffering of data (for example, databases on top of JFS file systems, where the database caches data in its own buffer cache; VMM also may have the same data in its persistent file cache). It is recommended that this option is used on systems such as these, especially if running HACMP; it can also be useful for systems where there is a lot of backup activity which causes frequent page replacement.

To make sure that the value set by maxperm is used in a strict way, and not only as recommended limits, set the strict_maxperm option:

```
# /usr/samples/kernel/vmtune -h 1
```

[2] For Oracle RDBMS servers, please see Chapter 17, “Oracle” on page 433.

The next example shows the settings that enable a large amount of programs to remain in memory and reduces the amount allowed for file caching. This scenario could be used on an Oracle database server:

```
# /usr/samples/kernel/vmtune -p 5 -P 20 -W32 -c2 -s1
```

This also sets the maxrandwrt[3] to 32*4 KB and numclust[4] to 2*16 KB and activates the sync_release_ilock parameter.

6.1.2.7 VMM memory load control facility

A process requires real-memory pages to execute. When a process references a virtual-memory page that is on disk, because it either has been paged-out or has never been read, the referenced page must be paged-in and, on average, one or more pages must be paged out (if replaced pages had been modified), creating I/O traffic and delaying the progress of the process.

The operating system attempts to steal real memory from pages that are unlikely to be referenced in the near future through the page-replacement algorithm. A successful page-replacement algorithm allows the operating system to keep enough processes active in memory to keep the CPU busy. But at some level of competition for memory, no pages are good candidates for paging out to disk because they will all be reused in the near future by the active set of processes.

Note

In AIX V4, memory load control is disabled by default on systems that have available memory frames that add up to greater than or equal to 128 MB.

6.1.2.8 Allocation and reclamation of paging space slots

The operating system supports three allocation methods for working storage, also referred to as paging-space slots, as follows:

- Deferred allocation
- Late allocation
- Early allocation

[3] Specifies a threshold in 4 KB pages for random writes to accumulate in RAM before these pages are synchronized to disk via a write-behind algorithm. This threshold is on a per file basis.
[4] Specifies the number of 16 KB clusters processed by write behind. This threshold is system wide.

Note

Paging-space slots are only released by process (not thread) termination or by the disclaim() system call. The slots are not released by the free() system call.

Deferred allocation algorithm

The default paging-space-slot-allocation method is the Deferred Page Space Allocation (DPSA) policy. The DPSA delays allocation of paging space until it is necessary to page out the page, which results in no wasted paging space allocation. This method can save huge amounts of paging space, which means disk space.

On some systems, paging space might not ever be needed, even if all the pages accessed have been touched. This situation is most common on systems with very large amount of RAM. However, this may result in over commitment of paging space in cases where more virtual memory than available RAM is accessed.

Late allocation algorithm

With the late allocation algorithm, a paging slot is allocated to a page of virtual memory only when that page is first read from or written into. The normal recommendation for paging-space size is at least twice the size of the system's real memory using this allocation algorithm. This was the default algorithm prior to AIX 4.3.2.

Early allocation algorithm

The early allocation algorithm causes the appropriate number of paging-space slots to be allocated at the time the virtual-memory address range is allocated. The early allocation algorithm is invoked by setting the environment variable PSALLOC=early for the programs that are to use the early allocation algorithm. The recommendation for paging space size for systems that use early allocation is at least four times the real memory size.

6.2 CPU

AIX scheduling uses a time-sharing, priority based scheduling algorithm. This algorithm favors threads that do not consume large amounts of the processor time, as the amount of processor time used by the thread is included in the priority recalculation equation. Fixed priority threads are allowed: the priority of these threads do not change regardless of the amount of processor time used.

Each new process is created with a single thread that has its parent process priority and contends for the CPU with the threads of other processes. The process owns the resources used in execution; the thread owns only its current state. A thread can be thought of as a low-overhead process. It is a dispatchable entity that requires fewer resources to create than a process. The fundamental dispatchable entity of the AIX V4 scheduler is the thread. Each process is made up of one or more threads. A thread is a single sequential flow of control. Multiple threads of control allow an application to overlap operations, such as reading from a terminal and writing to a file.

All processes in the system are created from other processes through the fork[5] mechanism. All processes are also descendants of the init process (process number 1) except the scheduler (process number 0).

6.2.1 Scheduler run queue management

A user thread within a process has a specified contention scope. If the contention scope is global, the thread contends for CPU time with all other threads in the system. The thread that is created when a process is created has global contention scope. If the contention scope is local, the thread contends with the other threads within the process to be the recipient of the process' share of CPU time.

The algorithm for determining which thread should be run next is called a scheduling policy.

6.2.1.1 Run queue and process states

The scheduler maintains a run queue of all of the threads that are ready to be dispatched.

[5] A process that forks creates a child process that is nearly a duplicate of the original parent process. Statistical information is reset and the child initially shares most of the virtual memory space with the parent process. The child process initially runs the same program as the parent process. The fork() or f_fork() system calls can be used to perform the fork operation.

All the dispatchable threads of a given priority occupy consecutive positions in the run queue.

The fundamental dispatchable entity of the AIX V4 scheduler is the thread. AIX V4 maintains 128 run queues. These run queues relate directly to the range of possible values (0 through 127) for the priority field for each thread. This method makes it easier for the scheduler to determine which thread is most favored to run. Without having to search a single large run queue, the scheduler consults a 128-bit mask where a bit is on to indicate the presence of a ready-to-run thread in the corresponding run queue.

The priority value of a thread changes rapidly and frequently because threads are always jumping from one run queue level to another. This is not true, however, for fixed-priority threads; the constant movement is due to the way that the scheduler recalculates priorities.

Starting with AIX V4.3.3, each CPU has its own 128-bit run queue. The run queue values reported in the performance tools will be the sum of all the threads in each run queue. If a thread becomes runnable because of an event on another CPU than the one in which the newly runnable thread had been running on, then this thread would only get dispatched immediately if there was an idle CPU. No preemption occurs until the processor's state can be examined (such as an interrupt on this thread's CPU).

If an environment variable called RT_GRQ exists, and is set to ON, it will cause this thread to be on a global run queue. In that case, the global run queue is searched to see which thread has the best priority. This can improve performance for threads that are running SCHED_OTHER and are interrupt driven. Threads that are running at fixed priority are placed on the global run queue if the FIXED_PRI setting is enabled with the `schedtune` command.

The average number of threads in the run queue can be seen in the first column of the `vmstat` (**r**) command output. If you divide this number by the number of CPUs, the result is the average number of threads that can be run on each CPU. If this value is greater than one, these threads must wait their turn for the CPU (the greater the number, the more likely it is that performance delays are noticed).

When a thread is moved to the end of the run queue (for example, when the thread has control at the end of a time slice), it is moved to a position after the last thread in the queue that has the same priority value.

The scheduler maintains processes in the multi-level run queue in the following states:

Idle Processes that are being created are in the idle state. This state is temporary until the fork mechanism is able to allocate all of the necessary resources for the creation of a new process. Once the new child process is created, it is placed in the active state.

Active This is the normal process state. Threads in the process can be running or ready-to-run.

Stopped Process has been stopped by SIGSTOP signal. Process can be restarted by SIGCONT signal. All threads are in the stopped state.

Swapped Process cannot run until scheduler makes it active again. All threads are in the swapped state.

Zombie Processes that have terminated with an exit system call are in the zombie state. Such processes have most of their resources freed. However, a small part of the process remains, such as the exit value that the parent process uses to determine why the child process died.

When a process terminates, and if the parent issues a wait system call (or catches the SIGCHLD signal), the exit status is returned to the parent. However, if the parent does not care about the child, the child will occupy a slot with the exit code until a parent acknowledges its termination (the resources that the child occupied are released upon termination before its exit status is delivered). If the parent no longer exists when a child process exits, the init process frees the remaining resources held by the child (the child will linger in the Zombie state until this is done).

6.2.1.2 Scheduling policy for threads

In AIX V4, the five possible values for thread scheduling policy are as follows:

SCHED_FIFO This is a non-preemptive scheduling scheme. After a thread with this policy is scheduled, it runs to completion unless it is blocked, it voluntarily yields control of the CPU, or a higher-priority thread becomes dispatchable. Only fixed-priority threads can have a SCHED_FIFO scheduling policy.

SCHED_RR The thread has a fixed priority. When a SCHED_RR thread has control at the end of the time slice, it moves to the tail of the queue of dispatchable threads of its priority. Only fixed-priority threads can have a SCHED_RR scheduling policy.

SCHED_OTHER This policy is defined by POSIX Standard 1003.4a as implementation-defined. In AIX V4, this policy is defined to be equivalent to SCHED_RR, except that it applies to threads with nonfixed priority. The recalculation of the running thread's priority value at each clock interrupt means that a thread may lose control because its priority value has risen above that of another dispatchable thread.

SCHED_FIFO2 The policy is the same as for SCHED_OTHER, except that it allows a thread which has slept for only a short amount of time to be put at the head of its run queue when it is awakened. This time period is the affinity limit (tunable with `schedtune -a`). This policy is only available beginning with AIX V4.3.3.

SCHED_FIFO3 A thread whose scheduling policy is set to SCHED_FIFO3 is always put at the head of a run queue. To prevent a thread belonging to SCHED_FIFO2 scheduling policy from being put ahead of SCHED_FIFO3, the run queue parameters are changed when a SCHED_FIFO3 thread is queued, so that no thread belonging to SCHED_FIFO2 will satisfy the criterion that enables it to join the head of the run queue. This policy is only available beginning with AIX V4.3.3.

6.2.1.3 Process and Thread Priority

The priority management tools manipulate process priority. In AIX V4, process priority is simply a precursor to thread priority. When the fork() subroutine is called, a process and a thread to run in it are created. The thread has the priority that would have been attributed to the process.

The kernel maintains a priority value (sometimes termed the scheduling priority) for each thread. The priority value is a positive integer and varies inversely with the importance of the associated thread. That is, a smaller priority value indicates a more important thread. When the scheduler is looking for a thread to dispatch, it chooses the dispatchable thread with the smallest priority value.

A thread can be fixed-priority or nonfixed priority. The priority value of a fixed-priority thread is constant, while the priority value of a nonfixed priority thread varies based on the sum of the minimum priority level for user threads (a constant 40), the thread's nice value (20 by default, optionally set by the `nice` or `renice` command), and its CPU-usage penalty.

The priority of a thread can be fixed at a certain value, which can have a priority value less than 40, if their priority is set (fixed) through the setpri() subroutine. These threads are immune to the scheduled recalculation algorithms. If their priority values are fixed to be less than 40, these threads will run and complete before any user threads can run. For example, a thread with a fixed value of 10 will run to completion before a thread with a fixed value of 15.

Users can apply the `nice` or `renice` command to make a thread's nonfixed priority less favorable (by adding more than the default nice value of 20 to the priority recalculation, thus increasing the value of the priority). The system manager can apply a negative nice value to a thread, thus giving it a better priority.

The nice value of a thread is set when the thread is created and is constant over the life of the thread, unless explicitly changed by the user.

6.2.1.4 CPU penalty for non-fixed processes

The CPU penalty is an integer that is calculated from the recent CPU usage of a thread. The recent CPU usage increases by 1 each time the thread is in control of the CPU at the end of a 10 ms clock tick, up to a maximum value of 120. Once per second, the recent CPU usage values for all threads are recalculated.

The result is the following:

- The priority of a nonfixed-priority thread becomes less favorable as its recent CPU usage increases and vice versa. This implies that, on average, the more time slices a thread has been allocated recently, the less likely it is that the thread will be allocated the next time slice.
- The priority of a nonfixed-priority thread becomes less favorable as its nice value increases, and vice versa.

6.2.1.5 Process priority values

The formula for calculating the priority value for a process is:

priority value = base priority + nice value + (CPU penalty based on recent CPU usage) or to be more precise:

Recalculated priority = p_nice + (C * R/32) where p_nice = base priority + nice value

The recent CPU usage value of a given thread is incremented by 1 each time that thread is in control of the CPU when the timer interrupt occurs (every 10 milliseconds). The recent CPU usage value is displayed as the C column in the `ps` command output. The maximum value of recent CPU usage is 120. The default algorithm calculates the CPU penalty by dividing recent CPU usage by 2. The CPU-penalty-to-recent-CPU-usage ratio is therefore 0.5. We will call this value R (the default is 16). The formula is as follows:

C = C_recent * R/32

Once a second, the default algorithm divides the recent CPU usage value of every thread by 2. The recent-CPU-usage-decay factor is therefore 0.5. We will call this value D (the default is 16). The formula is as follows:

C = C_old * D/32

For some users, the existing algorithm does not allow enough distinction between foreground and background processes. For example, ignoring other activity, if a system were running two compute-intensive user processes, one foreground (nice value = 20), and one background (nice value = 24) that started at the same time, the following sequence would occur:

The foreground process would be dispatched first. At the end of eight time slices (80 ms), its CPU penalty would be 4, which would make its priority value equal to that of the background process. The scheduling algorithm would cause the background process to be dispatched.

After two further time slices, the background process's CPU penalty would be 1, making its priority value one greater than that of the foreground process. The foreground process would be dispatched.

Another two time slices and the priority values of the processes would be equal again. The processes would continue to alternate every two time slices until the end of the second.

At the end of the second, the foreground process would have had 54 time slices and the background would have had 46. After the decay factor was applied, the recent CPU usage values would be 27 and 23. In the second of their competition, the foreground process would get only four more time slices than the background process.

Even if the background process had been started with the command `nice -n 20`, the distinction between foreground and background would be only slightly better. Although the scheduler stops counting time slices used after 120, this permits the CPU penalty to level off at 60, which is more than enough to offset the maximum nice value difference of 40.

To allow greater flexibility in prioritizing threads, the operating system permits user tuning of the ratio of CPU penalty to recent CPU usage (**R**) and the recent-CPU-usage-decay rate (**D**). Also, to have more of an impact on the priorities if the nice value was changed, the formula for calculating the priority value has been slightly changed in AIX V4.3.2 and later.

The formula now provides a mechanism such that threads whose nice values are the default of 20 will behave the same as they did previously, but when the nice value is less than or greater than the default, the priority will be affected more, due to two formulas that are being used.

Remember that p_nice = base priority + nice value. Now use the following:

If p_nice > 60, then x_nice = (p_nice * 2) - 60, else x_nice = p_nice.

What this means is that if the nice value is greater than 20, then the nice value has more of an impact on the priority value than if the nice value was less than or equal to 20. A new factor which we will call X is also used in the priority calculation.

X = (x_nice + 4)/64

The new priority calculation is as follows:

Recalculated priority = x_nice + (C * R/32 * X)

Note

As long as the nice value is the default, the formula of AIX V4.3.1 and earlier is still used.

6.2.1.6 Process mode switching

A user process undergoes a mode switch when it needs access to system resources. This is implemented through the system call interface or by interrupts, such as page faults. There are two modes:

- User mode
- Kernel mode

CPU time spent in user mode (application and shared libraries) is reflected as *user time* in the output of commands. CPU time spent in kernel mode is reflected as *system time* in the output of these commands.

User mode Programs executing in the user protection domain do not have access to the kernel or kernel data segments, except indirectly through the use of system calls. A program in this protection domain can only affect its own execution environment and executes in the process or unprivileged state.

Kernel mode Programs executing in this protection domain can affect the execution environments of all programs. The use of a system call by a user-mode process allows a kernel function to be called from user mode. Access to functions that directly or indirectly invoke system calls is typically provided by programming libraries, which provide access to operating system functions.

6.2.1.7 Process context switching

Mode switches should be differentiated from the context switches seen in the output of the `vmstat` (`cs` column) and `sar` (`cswch/s`) commands. A context switch occurs when the currently running thread is different from the previously running thread on that CPU.

The scheduler performs a context switch when any of the following occurs:

- A thread must wait for a resource (voluntarily), such as disk I/O, network I/O, sleep, or locks.
- A higher priority thread wakes up (involuntarily).
- The thread has used up its time slice (usually 10 ms).

Context switch time, system calls, device interrupts, NFS I/O, and any other activity in the kernel is considered as system time.

6.2.2 CPU tuning

Tuning how the system scheduler selects applications (threads running applications) is usually not necessary on currently available nodes if they run a mixed workload. But when necessary, it is mostly used to favor either interactive or batch users. Since I/O intensive applications becomes favored by the scheduler, on a overcommitted system with a mixed simultaneous workload, interactive users could be suffering, although the total throughput is high. Tuning is then done to reduce the batch throughput in favor of interactive response times.

It is also usually of no consequence for throughput, on the currently available nodes, to use `nice/renice` if you have thousands of processes (threads) and with hundreds to thousands of concurrent active users. The usage of `nice/renice` is mostly for systems with few users that have longer running jobs on the same node.

> **Important**
>
> Using `schedtune` as described in this section inappropriately can seriously harm your production environment. Be warned and take necessary steps to ensure that this will not happen. Know what you do before you do it and then do it right.

6.2.2.1 Modifying the scheduler priority calculation

Tuning is accomplished through two options of the `schedtune` command: -r and -d. Each option specifies a parameter that is an integer from 0 through 32. The parameters are applied by multiplying the recent CPU usage value by the parameter value and then dividing by 32 (shift right 5). The default *R* and *D* values are 16, which yields the same behavior as the original algorithm (D=R=16/32=.5). The new range of values permits a far wider spectrum of behaviors. *R* stands for the rate at which to accumulate CPU usage and *D* stands for the factor used to decay CPU usage.

Guidelines for R and D:

Smaller values of R narrow the priority range and make the nice value more of an impact on the priority. Larger values of R widen the priority range and make the nice value less of an impact on the priority.

Smaller values of D decay CPU usage at a faster rate and can cause CPU-intensive threads to be scheduled sooner. Larger values of D decay CPU usage at a slower rate and penalize CPU-intensive threads more (thus favoring interactive-type threads).

Tuning examples using R and D

Setting R to 0 (R=0, D=.5) would mean that the CPU penalty was always 0, making priority absolute. For example:

```
# /usr/samples/kernel/schedtune -r 0
```

No background process would get any CPU time unless there were no dispatchable foreground processes at all. The priority values of the threads would effectively be constant, although they would not technically be fixed-priority threads.

Setting R to 5 (R=.15625, D=.5) would mean that a foreground process would never have to compete with a background process started with the command `nice -n 20`. For example:

```
# /usr/samples/kernel/schedtune -r 5
```

The limit of 120 CPU time slices accumulated would mean that the maximum CPU penalty for the foreground process would be 18.

Setting R to 6 and D to 16 (R=.1875, D=.5) would mean that, if the background process were started with the command `nice -n 20`, it would be at least one second before the background process began to receive any CPU time. For example:

```
# /usr/samples/kernel/schedtune -r 6 -d 16
```

Foreground processes, however, would still be distinguishable on the basis of CPU usage. Long-running foreground processes that should probably be in the background would ultimately accumulate enough CPU usage to keep them from interfering with the true foreground.

Setting R to 32 and D to 32 as well (R=1, D=1) would mean that long-running threads would reach a *C* value of 120 and remain there, contending on the basis of their nice values. For example:

```
# /usr/samples/kernel/schedtune -r 32 -d 32
```

New threads would have priority, regardless of their nice value, until they had accumulated enough time slices to bring them within the priority value range of the existing threads.

6.2.2.2 Modifying the scheduler time slice

The length of the scheduler time slice can be modified with the `schedtune` command. To change the time slice, use the `schedtune -t` option, as in the following example:

```
# /usr/samples/kernel/schedtune -t 4
```

The value of -t is the number of ticks for the time slice and only fixed-priority threads (such as SCHED_RR) will use the nondefault time slice values.

A thread running with SCHED_OTHER scheduling policy can use the CPU for up to a full time slice (the default time slice being 1 clock tick) and a clock tick being 10 ms). However, if this thread is preempted by a clock interrupt or some other interrupt (such as disk or network adapter interrupt), this thread may not get its full time slice and will be put at the end of the priority-ordered run queues.

A thread running with SCHED_RR scheduling policy will always be guaranteed at least a full time slice. If it gets interrupted before its time slice has been completed, it will be put at the beginning of the run queue and will be allowed to run again.

In some situations, too much context switching (the `vmstat` and `sar` commands can indicate the number of context switches per second) is occurring and the overhead of dispatching threads can be more costly than allowing these threads to run for a longer time slice. In these cases, increasing the time slice might have a positive impact on performance on fixed-priority threads.

6.2.2.3 Tuning the NICE value for individual processes and threads

To alter the nice value for a process, two commands can be used: `nice` and `renice`. Any user can run a command at a less-favorable-than-normal priority by using the `nice` command. The `renice` command alters the nice value of one or more processes that are already running. The processes are identified either by process ID, process group ID, or the name of the user who owns the processes.

Note

Only the *root* user can use the `nice` and `renice` command to run applications at a more-favorable-than-normal priority.

The `renice` command also has the following two limitations:

- Only processes owned by that user ID can be specified.
- The priority of the process cannot be increased, not even to return the process to the default priority after making its priority less favorable with the `renice` command.

Using the nice command

With the `nice` command, the user specifies a value to be added to or subtracted from the standard nice value. The modified nice value is used for the process that runs the specified command. The priority of the process is still non-fixed; that is, the priority value is still recalculated periodically based on the CPU usage, nice value, and minimum user-process-priority value.

The standard nice value of a foreground process is 20; the standard nice value of a background process is 24. The nice value is added to the minimum user-process-priority level (40) to obtain the initial priority value of the process. For example, the command:

```
# nice -n 5 application_program
```

causes the `application_program` command to be run with a nice value of 25 (instead of 20), resulting in a base priority value of 65 (before any additions for recent CPU use). If we were using *root* login, we could have run the `application_program` command at a more favorable priority with:

```
# nice -n -5 application_program
```

If we were not using *root* login and issued that `nice` command, the `application_program` command would still be run, but at the standard nice value of 20, and the `nice` command would not issue any error message.

Using the renice command

For AIX V4, the syntax of the `renice` command has been changed to complement the alternative syntax of the `nice` command, which uses the -n flag to identify the nice-value increment.

The `renice` command can not be used on fixed-priority processes. A user can specify a value to be added to, but not subtracted from the priority of one or more running processes. The modification is done to the nice values of the processes. The priority of these processes is still non-fixed. Only the root user can use the `renice` command to alter the priority value within the range of -20 to 20, or subtract from the priority of one or more running processes.

To continue our example, we will use the `renice` command to alter the nice value of the `application_program` process that we started with `nice`.

```
# renice -n -5 7569
```

By reducing the nice value of the process running `application_program` (with process id 7569) by 5, it will be 15, if it previously was 20.

6.3 I/O

The I/O subsystem is the slowest part in a standalone RS/6000 and, for practical purposes, in a SP system as well. It does not matter if the DASD[6] (more commonly know as disk, both fixed and removable) are SCSI, SSA, FC connected, ESS, or RAID. At one time or another the reading and writing to the disks, in a large multi-user SMP SP node, will be a constraining bottleneck for throughput. Only solid state “disks” or disks with true read/write caching will be fast enough to feed data to and from memory, but fetching data from memory will still be much faster.

It is for this reason that applications such as Oracle VLM RDBMS, with medium sized databases, that will fit in a couple of GB memory, will run much faster, because the data can be accessed directly from memory instead of using a complicated caching mechanism. But since memory based storage is dependent on continuous electricity, it can not be relied upon to store data over a long period of time, so disk storage will still need to be used. (For large database systems with TB of data, this is not a solution anyway.)

To do proper performance tuning for disk I/O, it is necessary to understand the path that data has to travel from the disk until it reaches memory and vice versa. In this section, we will focus on LVM and VMM tuning.

6.3.1 Logical volume manager

The Logical Volume Manager (LVM) controls disk resources by mapping data between a more simple and flexible logical view of storage space and the actual physical disks. The LVM does this using a layer of device driver code that runs above traditional disk device drivers. The LVM consists of the logical volume device driver (LVDD) and the LVM subroutine interface library. The logical volume device driver (LVDD) is a pseudo-device driver that manages and processes all I/O. It translates logical addresses into physical addresses and sends I/O requests to specific device drivers.

[6] Direct access storage devices.

6.3.2 Data transfer path to and from disk

Conceptually the data transfer path from disk to memory is described in Table 19.

Table 19. Simplified transfer path of data from disk to memory

Operation	Performance implications
Determining position of data blocks on disk.	LVM and disk device driver operations. The overhead depends on the type of reading that is performed.
Determine position for data on disk and disk arm seeks this position on the disk.	Mainly seek time for the disk.
Read data from disk and transfer from disk to disk controller.	Read transfer rate of disk in MB/s.
Transfer data from disk controller to disk adapter using the disk bus.	Transfer rate of disk bus in MB/s.
Transfer data from disk adapter to memory using the adapter bus.	Adapter bus transfer rate (using DMA).
Transfer data from memory to registers using intermediary processor caches.	Memory transfer logic that is CPU dependent, but never a constraining factor in data transfer from disk.

Discussions of disk, logical volume and file system performance sometimes lead to the conclusion that the more drives you have on your system, the better the disk I/O performance. This is not always true because there is a limit to the amount of data that can be handled by a disk adapter. The disk adapter can also become a bottleneck. If all your disk drives are on one disk adapter, and your hot file systems are on separate physical volumes, you might benefit from using multiple disk adapters. Performance improvement will depend on the type of access.

The major performance issue for disk drives is usually application-related; that is, whether large numbers of small accesses will be made (random), or smaller numbers of large accesses (sequential). For random access, performance will generally be better using larger numbers of smaller capacity drives. The opposite situation exists for sequential access (use faster drives or use striping with larger number of drives).

A disk consists of a set of flat, circular rotating platters. Each platter has one or two sides on which data is stored. Platters are read by a set of nonrotating, but positionable, read or read/write heads that move together as a unit. The terms in Table 20 are used when discussing disk device block operations.

Table 20. Terms used in describing disk device block operations

Term	Description
sector	An addressable subdivision of a track used to record one block of a program or data. On a disk, this is a contiguous, fixed-size block. Every sector of every DASD is exactly 512 bytes.
track	A circular path on the surface of a disk on which information is recorded and from which recorded information is read; a contiguous set of sectors. A track corresponds to the surface area of a single platter swept out by a single head while the head remains stationary.
head	A head is a positionable entity that can read and write data from a given track located on one side of a platter. Usually a DASD has a small set of heads that move from track to track as a unit.
cylinder	The tracks of a DASD that can be accessed without repositioning the heads. If a DASD has n number of vertically aligned heads, a cylinder has n number of vertically aligned tracks.

6.3.2.1 Disk access times

The three components that make up the access time of a disk are described in Table 21.

Table 21. Latencies for disk access times

Latency	Description
Seek	A seek is the physical movement of the head at the end of the disk arm from one track to another. The time for a seek is the necessary time for the disk arm to accelerate, to travel over the tracks to be skipped, to decelerate, and to settle down and wait for the vibrations to stop while hovering over the target track. The total time the seeks take is variable. The average seek time is used to measure the disk capabilities.
Rotational	This is the time that the disk arm has to wait while the disk is rotating underneath until the target sector approaches. Rotational latency is, for all practical purposes (except sequential reading), a random function with values uniformly between zero and the time required for a full revolution of the disk. The average rotational latency is taken as the time of a half revolution. To determine the average latency, you must know the number of revolutions per minute (RPM) of the drive. By converting the revolutions per minutes to revolutions per second and dividing by two, we get the average rotational latency.

Latency	Description
Transfer	The data transfer time is determined by the time it takes for the requested data block to move through the read/write arm. It is linear with respect to the block size. The average disk access time is the sum of the averages for seek time and rotational latency plus the data transfer time (normally given for a 512-byte block). The average disk access time generally overestimates the time necessary to access a disk; typical disk access time is 70 percent of the average.

6.3.3 Disk types

Currently, SP nodes supports both SCSI and SSA connected disks. The disks can be both IBM and non-IBM. However, when using IBM manufactured disks that IBM sells, the firmware and microcode is usually not the same as for IBM OEM disk that is sold by other companies. This has implications on how the disks (especially SCSI) will be initially configured in AIX by the config manager.

IBM disks will have default settings in ODM PdAT that states how certain characteristics of the SCSI interface can be used for that particular disk. IBM OEM and other manufacturers disk will be configured as other disks. For these disks, it is important to check if and how they support the SCSI command tag queuing feature.

6.3.3.1 SCSI

SCSI (Small Computer System Interface), an ANSI standard, supports eight devices (one for the server adapter) via a parallel bus. Original SCSI-1 was rated at 3.5 MB/s, and the latest was up to 5 MB/sec. SCSI-2 became an ANSI standard in 1974 and added performance features: SCSI fast and SCSI wide. More devices could be added and the data transfer rate was also increased. SCSI-2 has a top speed of 20 MB/s. SCSI-3 includes four different types of interconnect technologies: SPI, SSA, FCS, and Firewire. SPI (SCSI-3 Parallel Interface) is also known as Ultra SCSI. The 16-bit UltraSCSI has a top speed of 40 MB/s. SCSI (1,2,and Ultra) are arbitrated bus architectures. To transfer data, a SCSI device has to gain control over the SCSI bus. Thus, only one SCSI device can be transferring data at a time. Once a device has arbitrated for access, it can then submit the data. When multiple SCSI devices try to access the bus at the same time, the port with the highest priority will get the arbitration. Under peak loads, ports with the lowest priority can be starved for service.

Command tag queuing for non-IBM SCSI disks

SCSI Command tag queuing refers to queuing multiple commands to a SCSI device. A command to a file system on a SCSI disk or to the raw disk causes the SCSI disk driver to generate one or more transactions. These transactions are then sent to the SCSI adapter driver, which sends them to the SCSI adapter, which in turn sends them to the disk itself. The following `chdev` command sets the queue depth to 3:

```
# chdev -l hdisk# -a queue_depth=3
```

Queuing to the SCSI device can improve performance, because the device itself determines the most efficient way to order and process commands. The default queue depth for IBM SCSI disks are 3.

Note

For non-IBM RAID solutions, the queue depth could be increased additionally by using the number of disks multiplied with the queue depth for each disk. For example: on a five disk RAID, where each disk performs well with a queue depth of 3, we could set the queue_depth for the RAID disk to 15 (5*3).

SCSI devices that support command tag queuing can be divided into two classes: those that clear their queues on error and those that do not. Devices that do not clear their queues on error resume processing of queued commands when the error condition is cleared (typically by receiving the next command). Devices that do clear their queues flush all commands currently outstanding. When the q_err attribute for the disk is yes, the SCSI disk clears all transactions from its command queue when an error occurs. If this attribute is no, the transactions remain on the queue. The following example shows how to set this attribute:

```
# chdev -l hdisk# -a q_err=yes
```

When the SCSI disk drive does not support the Qerr bit, the clr_q attribute can be used to indicate to the system how the disk handles its queue of transactions if an error occurs. When yes, this attribute indicates that the SCSI disk drive clears its queue when an error occurs. When set to no, this attribute indicates that the disk drive does not clear its command queue when an error occurs. The system uses this attribute only if the system determines that the drive does not support the Qerr bit. If the SCSI disk supports the Qerr bit, the system uses the value of the Use Qerr Bit attribute to determine how to configure the drive. Unless the drive manufacturer indicates otherwise, set this attribute to no.

```
# chdev -l hdisk# -a clr_q=yes
```

The Queuing type attribute must be set to either simple or ordered. Do not change this attribute without understanding the SCSI-2 specification for command tag queuing because some settings can degrade performance. For drives that support command tag queuing, this value should not exceed the drive's queue size. You may need to determine the best value by trial and error. The ordered value indicates that the system can queue multiple transactions to the disk and that the drive processes the transactions in the order that it receives them. However, the system can reorder transactions even if the ordered value is selected. The simple value indicates that the system can queue multiple transactions at the drive and that the drive can process the transactions in a more efficient order than that which they arrived in. The following example enables command queue tagging and sets the queue type to simple:

```
# chdev -l hdisk# -a type=simple
```

6.3.3.2 SSA

SSA is part of the SCSI-3 standard and is enhanced to support fault tolerance and high-performance loops with a large number of devices and does not require arbitration. An SSA adapter has four full-duplex ports. Multiple devices can transmit data simultaneously by a characteristic called spatial reuse. Spatial reuse allows for higher throughput than indicated by the interface bandwidth. Because each serial point-to-point link is independent, multiple data transfers can occur at the same time on the same loop. During periods of heavy data traffic, SSA's fairness algorithm ensures that each device on the loop gets its fair share of time.

Disk placement within a loop

The SSA subsystem uses a fairness algorithm to guarantee each port on a node will be allowed to originate a pre-programmed amount of frames. This is only of interest if a node wishes to originate a frame and at the same time must forward a frame to the next node. Generally, if a node needs to forward a frame, the hardware will route the frame before it allows the node to originate a new frame. Thus, you will want to place the disks with the lowest I/O nearest the adapter, and the disks with the highest I/O furthest from the adapter. The increase in latency, for placing disks further from the adapter, is only about 5 microseconds per hop, so even if a disk is 20 hops from the adapter, the increased latency is only 0.1 ms. This algorithm is loaded in firmware, can be different for each node, could change in the future, and cannot be changed by the system administrator. The SSA loop must be heavily loaded for disk placement within the loop to matter. To determine the disk placements within loops, use simple commands as shown in the following example:

```
# for p in $(ssadisk -a ssa0 -P);do ssaconn -l $p -a ssa0;done|sort -k5
pdisk2          ssa0     -     -     0     3
pdisk3          ssa0     -     -     1     2
pdisk1          ssa0     -     -     2     1
pdisk0          ssa0     -     -     3     0
```

The above example shows us that pdisk2 is closest to port B1 and pdisk0 is closest to port B2. The output from the `ssaconn` is interpreted as follows:

```
                                    # of hops from adapter port

Logical disk           Adapter     A1       A2       B1       B2
```

In a system with two adapters in a loop, and unbalanced I/O between the adapters, changing the primary adapter for the pdisk and hdisk can improve performance. Note that you can also physically rearrange the disks to balance the load. The volume group must be varied off to make the change to the hdisk.

SSA RAID

RAID (Redundant Array of Independent Disks) allows a set of disk drives to be managed as a group. Each RAID level employs different data management algorithms:

- *RAID 0*: Data is striped in units of 512-byte blocks across consecutive physical drives. The first segment of logical blocks is located on the first disk in the array, the second segment is located on the second disk, and so on. Independent paths go to the drives, and spreading of segment-length portions of data is repeated across the entire array of disks. The number of drives, as well as the number of drive groups, is scalable. Drive spindles may be synchronized, but synchronization is not required.

  For current SSA implementation, there is no data protection, and a single drive failure results in data loss. Only a single initiator (one adapter) per loop configuration is allowed and the Fast-Write Cache (FWC) is supported. There can be from 2 to 16 disks per array, but all members and spares must be on the same loop.

- *RAID 1*: The array is split into two groups. The first set of drives stores original data (striped one segment at a time); the second stores the mirror image of the first. Independent data paths go to the drives, and spindles may be synchronized.

For current SSA implementation, two initiators (adapters) per loop are allowed and the Fast-Write Cache option is supported in both one or two initiator configurations. There can be only two drives per array and all members and spares must be on the same loop.

- *RAID 5* can read and write data and parity across all disks. The parity segments rotate among the drives. By eliminating the dedicated parity disk, the performance of the array is increased and another potential point of failure is removed. Independent data paths go to each drive, and synchronization of the spindles is suggested but not required.

  For current SSA implementation, from 3 to 16 disks per array are supported. However the number of initiators, the use of Fast-Write Cache (FWC), and if member and spare disks are allowed to span more than one loop, are dependent on the type of adapters used in a loop.

- *RAID 10* or Enhanced RAID 1 is RAID 1 but with striping as well.

  For current SSA implementation, two initiators (adapters) per loop are allowed and the Fast-Write Cache option is supported in both one or two initiator configurations. There can be from 4 to 16 disks per array (even numbers only). Member disks are designated as either Primary or Secondary (copy). There is no difference in the use of the disks by this designation, except when using split site operations.

For commercial production, RAID 5 or 10 with a maximum size Fast-Write Cache (FWC) is recommended. But as everything else with performance tuning, to make a design decision, the characteristics of the workload must be known.

max_coalesce setting

The SSA RAID parameter max_coalesce is the maximum number of bytes which the SSA disk device driver attempts to transfer to or from an SSA logical disk in a single operation. The default value is appropriate for most environments. For applications that perform very long sequential write operations, there are performance benefits in writing data in blocks of 64 KB times the number of disks in the array minus one (these are known as full-stride writes times the number of disks in the array minus one, or to some multiple of this number). Thus, the formula for calculating the optimum value for full-stride writes would be:

(N-1)*64K

or, to make it simple, just insert the value for *N-1* (since the hex value 0x10000 is 64K):

0x(N-1)0000

In the case of a RAID 5 array with eight disks, of which one is logically the parity drive, the value for max_coalesce would be 448K ((8-1)*64K) and 0x70000 in hex. The maximum value for max_coalesce is 0x200000 (2MB).

The current value of max_coalesce can be ascertained via the `lsattr` command. For example:

```
# lsattr -El hdisk6 -a max_coalesce
max_coalesce 0x20000 Maximum coalesced operation True
```

In general, increasing max_coalesce increases the chance that composite I/O operations for contiguous data are physically executed as single operations. This can clearly give performance benefits where the parameter can be set so as to maximize full-stride write operations.

To calculate the hexadecimal value, you could use the `bc` command and create a script like the one shown here:

```
#!/bin/ksh
# bc.calc
# B.Roden

i=${1:-20000}
op=${2:-'/1024'}
ibase=${3:-16} # Input base 10=dec, 16=hex, 8=oct
obase=${4:-10} # Output base 10=dec, 16=hex, 8=oct

print "scale=2\n$(print "obase=$obase\nibase=$ibase\n$i"|bc)$op"|bc
```

To check the decimal value of max_coalesce for the disk in the above example, run the following commands:

```
# bc.calc $(lsattr -El hdisk6 -a max_coalesce -F value|cut -c3-)
128.00

# bc.calc '(128*1024)' ' ' 10 16
20000
```

max_coalesce can be modified by use of the `chdev` command. However, the volume group must be varied off and on to make the change effective. For example:

```
# varyoffvg vg99
# chdev  -l hdisk6 -a max_coalesce=0x70000
hdisk6 changed
# varyonvg vg99
```

queue_depth setting

Indicates the maximum number of concurrent I/O operations which can be sent to the disk. The default value is correct for normal operating conditions.

The maximum value is 200 and increasing the queue_depth setting for large disk systems could be done.

The current value of queue_depth can be ascertained via the `lsattr` command. For example:

```
# lsattr -El hdisk6 -a queue_depth
queue_depth 32 Queue depth True
```

queue_depth can be modified by use of the `chdev` command. However the volume group must be varied off and on to make the change effective. For example:

```
# varyoffvg vg99
# chdev  -l hdisk6 -a queue_depth=200
hdisk6 changed
# varyonvg vg99
```

write_queue_mod setting

The write_queue_mod parameter alters the way in which write commands are queued to SSA logical disks. The default value is 0 for all SSA logical disks that do not use the fast-write cache; with this setting the SSA disk device driver maintains a single seek-ordered queue of queue_depth operations on the disk. Reads and writes are queued together in this mode.

If write_queue_mod is set to a non-zero value, the SSA disk device driver maintains two separate seek-ordered queues, one for reads and one for writes. In this mode, the device driver issues up to queue_depth read commands and up to write_queue_mod write commands to the logical disk.

This facility is provided because, in some environments, it may be beneficial to hold back `write` commands in the device driver so that they may be coalesced into larger operations, which may be handled as full-stride writes by the RAID software within the adapter.

Changing the write_queue_mod setting is unlikely to be useful unless a large percentage of the workload to a RAID 5 device is composed of sequential `write` operations.

The current value of write_queue_mod can be ascertained via the `lsattr` command. For example:

```
# lsattr -El hdisk6 -a write_queue_mod
write_queue_mod 0 Write queue depth modifier True
```

write_queue_mod can be modified by use of the `chdev` command. However, the volume group must be varied off and on to make the change effective. For example:

```
# varyoffvg vg99
# chdev  -l hdisk6 -a write_queue_mod=0
hdisk6 changed
# varyonvg vg99
```

scat_gat_pages setting

The scat_gat_pages attribute for the SSA adapter controls the size of the scatter-gather table used by the device driver. This table is used to control I/O operations sent to the adapter. If it is too small, you may experience degraded adapter I/O performance when under heavy load. If it is too large, it will needlessly consume kernel memory, which can also affect system performance. The size of the table is specified in 4 KB pages. If you have many disks attached to the adapter, which are used intensively, then consider setting the table size quite large. If the anticipated I/O load to the adapter is modest, then consider reducing the table size to conserve memory. Setting it too small will cause adapter I/O performance to degrade, while setting it too large will just use up memory.

The current value of scat_gat_pages can be ascertained via the `lsattr` command. For example:

```
# lsattr -El ssa0 -a scat_gat_pages
scat_gat_pages 35 Pages allocated for scatter/gather True
```

scat_gat_pages can be modified by use of the `chdev` command. However, the volume group must be varied off and on to make the change effective. For example:

```
# varyoffvg vg99
# chdev  -l ssa0 -a scat_gat_pages=35
ssa0 changed
# varyonvg vg99
```

MCA (Micro channel) SSA adapters do not have this attribute.

dma_mem setting

The dma_mem attribute for the SSA adapter sets the total bus address space, which is available for DMA transfers for commands which are active or queued on SSA devices accessed via this adapter. The default value is sufficient for normal circumstances. The dma_mem attribute should only be changed if large numbers of devices supporting long queues need to be kept active with I/O. Increasing the value may result in increased system performance if the current value is causing I/O to be queued in the adapter device driver.

If the system attempts to initiate more concurrent I/O requests than can be serviced with the available DMA space, then some I/O requests will be queued in the adapter device driver. If this happens, then the total system performance may be affected, depending upon the workload.

Increasing the value may result in some adapter cards not being configured if the total amount of bus memory space requested by all of the adapter cards on a particular bus is greater than what is available on the bus.

The current value of dma_mem can be ascertained via the `lsattr` command. For example:

```
# lsattr -El ssa0 -a dma_mem
dma_mem 0x400000 DMA bus memory length
```

dma_mem can be modified by use of the `chdev` command. However, the volume group must be varied off and on to make the change effective. For example:

```
# varyoffvg vg99
# chdev  -l ssa0 -a dma_mem=0x400000
ssa0 changed
# varyonvg vg99
```

This is a PCI SSA adapter attribute. The MCA attribute is dma_bus_mem.

6.3.4 Improving read and write performance (VMM)

The AIX operating system has several enhancing mechanisms for both reading and writing data to disk to minimize the time an application spends waiting for I/O. To enhance the read transfer rate for applications that read data sequentially, AIX utilizes read-ahead for JFS files. To enhance the write transfer rate for applications, AIX uses write-behind so that the application does not have to wait for the completion of the physical I/O of each write operation.

6.3.4.1 Sequential-access read ahead

The VMM tries to anticipate the future need for pages of a sequential file by observing the pattern in which a program is accessing the file. When the program accesses two successive pages of the file, the VMM assumes that the program will continue to access the file sequentially, and the VMM schedules additional sequential reads of the file. These reads are overlapped with the program processing, and will make the data available to the program sooner than if the VMM had waited for the program to access the next page before initiating the I/O. The number of pages to be read ahead is determined by two VMM thresholds:

minpgahead
: Number of pages read ahead when the VMM first detects the sequential access pattern. If the program continues to access the file sequentially, the next read ahead will be for 2 times minpgahead, the next for 4 times minpgahead, and so on until the number of pages reaches maxpgahead.

maxpgahead
: Maximum number of pages the VMM will read ahead in a sequential file.

If the program deviates from the sequential-access pattern and accesses a page of the file out of order, sequential read ahead is terminated. It will be resumed with minpgahead pages if the VMM detects a resumption of sequential access by the program. The values of minpgahead and maxpgahead can be set with the `vmtune` command.

Values of maxpgahead greater than 16 (read ahead of more than 64 KB) exceed the capabilities of some disk device drivers. In such cases, the read size stays at 64 KB. Higher values of maxpgahead can be used in systems where the sequential performance of striped logical volumes is of paramount importance.

A minpgahead value of 0 effectively defeats the mechanism. This can adversely affect performance. However, it can be useful in some cases where I/O is random, but the size of the I/Os cause the VMM's read-ahead algorithm to take effect.

NFS and the VMM have been changed, starting with AIX V4.3.3, to automatically turn off VMM read-ahead if it is operating on a locked file. On these types of files, read-ahead pages are typically flushed by NFS so that reading ahead is not helpful.

The buildup of the read-ahead value from minpgahead to maxpgahead is quick enough that for most file sizes there is no advantage to increasing minpgahead.

Tuning sequential-access read ahead

It is rare that tuning the sequential read-ahead feature (or turning it off) will improve performance.

The minpgahead and maxpgahead values can be changed by using options -r and -R in the `vmtune` command, as in the following example:

```
# /usr/samples/kernel/vmtune -r 0 -R 16
```

The values should be from the set: 0, 1, 2, 4, 8, 16, and so on. The use of other values may have adverse performance or functional effects. Values should be powers of 2 because of the doubling algorithm of the VMM.

6.3.4.2 Write behind

To increase write performance, limit the number of dirty file pages in memory, reduce system overhead, and minimize disk fragmentation, the file system divides each file into 16 KB partitions. The pages of a given partition are not written to disk until the program writes the first byte of the next 16 KB partition. At that point, the file system forces the four dirty pages of the first partition to be written to disk. The pages of data remain in memory until their frames are reused, at which point no additional I/O is required. If a program accesses any of the pages before their frames are reused, no I/O is required.

If a large number of dirty file pages remain in memory and do not get reused, the syncd daemon writes them to disk, which might result in abnormal disk utilization. To distribute the I/O activity more efficiently across the workload, write-behind can be turned on to tell the system how many pages to keep in memory before writing them to disk. The write-behind threshold is on a per-file basis, which causes pages to be written to disk before the sync daemon runs. The I/O is spread more evenly throughout the workload.

There are two types of write-behind: *sequential* and *random*.

Sequential write-behind

By default, a file is partitioned into 16 KB partitions or four pages. Each of these partitions is called a cluster. If all four pages of this cluster are dirty, then as soon as a page in the next partition is modified, the four dirty pages of this cluster are scheduled to go to disk. Without this feature, pages would remain in memory until the syncd daemon runs, which could cause I/O bottlenecks and fragmentation of the file.

Tuning sequential write-behind

The number of clusters that the VMM uses as a threshold is tunable. The default is one cluster. You can delay write-behind by increasing the numclust parameter using the `vmtune` command, as in the following example, that increases the number of write-behind clusters to 32 (32*16384 => 1/2 MB):

```
# /usr/samples/kernel/vmtune -c 32
```

Random write-behind

There may be applications that do a lot of random I/O, that is, the I/O pattern does not meet the requirements of the write-behind algorithm and all the pages stay resident in memory until the syncd daemon runs. If the application has modified many pages in memory, this could cause a very large number of pages to be written to disk when the syncd daemon issues a sync() call.

The write-behind feature provides a mechanism such that when the number of dirty pages in memory for a given file exceeds a defined threshold, these pages are then scheduled to be written to disk.

Tuning random write-behind

This threshold can be tuned by using the -W option of the `vmtune` command. The parameter to tune is maxrandwrt; the default value is 0, indicating that random write-behind is disabled. Increasing this value to 128 indicates that once 128 memory-resident pages of a file are dirty; any subsequent dirty pages are scheduled to be written to the disk. The first set of pages will be flushed after a sync() call.

The following example increases the number of write-behind clusters to 128 (128*4096 => 1/2 MB)

```
# /usr/samples/kernel/vmtune -W 128
```

6.3.5 Logical Volume Device Driver

The Logical Volume Device Driver (LVDD) is a pseudo-device driver that operates on logical volumes through the /dev/lv# special file. Like the physical disk device driver, this pseudo-device driver provides character and block entry points with compatible arguments. Each volume group has an entry in the kernel device switch table. Each entry contains entry points for the device driver and a pointer to the volume group data structure. The logical volumes of a volume group are distinguished by their minor numbers.

6.3.5.1 Scheduler layer

The scheduler layer schedules physical requests for logical operations and handles mirroring and the Mirror Write Consistency (MWC) cache. For each logical request, the scheduler layer schedules one or more physical requests. These requests involve translating logical addresses to physical addresses, handling mirroring, and calling the LVDD physical layer with a list of physical requests.

6.3.5.2 Physical layer

The physical layer of the LVDD handles startup and termination of the physical request. The physical layer calls a physical disk device driver's ddstrategy entry point with a list of buf structures linked together. In turn, the physical layer is called by the iodone kernel service when each physical request is completed. This layer also performs bad-block relocation and detection/correction of bad blocks, when necessary.

6.3.5.3 Bad blocks

If a logical volume is mirrored, a newly detected bad block is fixed by relocating that block. A good mirror is read and then the block is relocated using data from the good mirror. With mirroring, the user does not need to know when bad blocks are found. However, the physical disk device driver does log permanent I/O errors, so the user can determine the rate of media surface errors.

Detecting bad blocks

When a bad block is detected during I/O, the physical disk device driver sets the error fields in the buf structure to indicate that there was a media surface error. The physical layer of the LVDD then initiates any bad-block processing that must be done.

If the operation was a nonmirrored read, the block is not relocated, because the data in the relocated block is not initialized until a write is performed to the block. To support this delayed relocation, an entry for the bad block is put into the LVDD defects directory and into the bad-block directory on disk. These entries contain no relocated block addresses and the status for the block is set to indicate that relocation is desired.

On each I/O request, the physical layer checks whether there are any bad blocks in the request. If the request is a write and contains a block that is in a relocation-desired state, the request is sent to the physical disk device driver with safe hardware relocation requested. If the request is a read, a read of the known defective block is attempted.

If the operation was a read operation in a mirrored LV, a request to read one of the other mirrors is initiated. If the second read is successful, then the read is turned into a write request and the physical disk device driver is called with safe hardware relocation specified to fix the bad mirror.

If the hardware relocation fails or the device does not support safe hardware relocation, the physical layer of the LVDD attempts software relocation. At the end of each volume is a reserved area used by the LVDD as a pool of relocation blocks. When a bad block is detected and the disk device driver is unable to relocate the block, the LVDD picks the next unused block in the relocation pool and writes to this new location. A new entry is added to the LVDD defects directory in memory (and to the bad-block directory on disk) that maps the bad-block address to the new relocation block address. Any subsequent I/O requests to the bad-block address are routed to the relocation address.

Relocating bad blocks

The physical layer of the LVDD checks each physical request to see if there are any known software-relocated bad blocks in the request. The LVDD determines if a request contains known software-relocated bad blocks by hashing the physical address. Then a hash chain of the LVDD defects directory is searched to see if any bad-block entries are in the address range of the request.

If bad blocks exist in a physical request, the request is split into pieces. The first piece contains any blocks up to the relocated block. The second piece contains the relocated block (the relocated address is specified in the bad-block entry) of the defects directory. The third piece contains any blocks after the relocated block to the end of the request or to the next relocated block. These separate pieces are processed sequentially until the entire request has been satisfied.

Note

By default, AIX will not allow the execution of the intermediate-level data relocation on non-IBM drives. On IBM (not IBM OEM) drives, the AIX LVM relies on the guarantee that there are reserved areas for possible hardware relocation (typically 256 blocks per disk). However, on non-IBM drives, AIX LVM cannot predict what spare blocks may or may not exist for possible relocation. So, assuming the worst, AIX LVM only allows the internal disk relocation and the top layer software relocation to occur on non-IBM drives.

6.3.6 Performance implications of disk mirroring

Disk mirroring is mostly used to reduce impact of disk failure on running applications. Systems that need large storage capacity use either ESS (or equivalent product), RAID solutions or AIX managed SSA disks in drawers and racks. And since disks fail to work sometimes, if many disks are used (a couple of hundred), a couple of disks will fail during each production year (sometimes during a month). So in this section, we will discuss how to tune the I/O system when using mirroring.

6.3.6.1 Mirror write consistency

The LVM always ensures data consistency among mirrored copies of a logical volume during normal I/O processing. For every write to a logical volume, the LVM generates a write request for every mirror copy. A problem arises if the system crashes in the middle of processing a mirrored write (before all copies are written). If mirror write consistency recovery is requested for a logical volume, the LVM keeps additional information to allow recovery of these inconsistent mirrors. Mirror Write Consistency recovery should be performed for most mirrored logical volumes. Logical volumes, such as the page space that do not use the existing data when the volume group is re-varied on, do not need this protection. Although an `lslv` command will usually show MWC to be on for nonmirrored logical volumes, no actual processing is incurred unless the COPIES value is greater than one.

Mirror write consistency record

The Mirror Write Consistency (MWC) record consists of one sector. It identifies which logical partitions may be inconsistent if the system is not shut down correctly. When the volume group is varied back on-line, this information is used to make the logical partitions consistent again.

Mirror write consistency check

The Mirror Write Consistency Check (MWCC) is a method for tracking the last 62 writes to a mirrored logical volume.[7] MWCC only makes mirrors consistent when the volume group is varied back online after a crash by examining the last 62 writes to mirrors and picking one mirror and propagating that data to the other mirrors. MWCC does not keep track of the latest data; it only keeps track of LTGs currently being written. Therefore, MWC does not guarantee that the latest data will be propagated to all the mirrors; it is the application above LVM that has to determine the validity of the data after a crash.

This source that is selected for propagation is important to parallel mirrored systems. In sequentially mirrored systems, the source is always the primary disk. If that disk fails to respond, the next disk in the sequential ordering will be picked as the source copy. There is a chance that the mirror picked as source to correct the other mirrors was not the one that received the latest write before the system crashed. Thus, the write that may have completed on one copy and incomplete on another mirror would be lost.

AIX does not guarantee that the absolute, latest write request completed before a crash will be there after the system reboot. But, AIX will guarantee that the parallel mirrors will be consistent with each other.

If the mirrors are consistent with each other, then the user will be able to realize which writes were considered successful before the system crashed and which writes will be retried. The point here is not data accuracy, but data consistency. The use of the primary mirror copy as the source disk is the

[7] It is actually journaling that a Write request is active in a Logical Track Group (LTG) (32 4 KB pages or 128 KB)

basic reason that sequential mirroring is offered. Not only is data consistency guaranteed with MWC, but the use of the primary mirror as the source disk increases the chance that all the copies have the latest write that occurred before the mirrored system crashed.

Tuning mirror write consistency

MWC can take two writes just to perform one write. To turn off MWC for a logical volume, run the `chlv` command, as in the following example for logical volume `lv99`:

```
# chlv -w n lv99
```

If MWC is turned off, the `autovaryon` on the volume group must also be turned off, as in the following example for volume group `vg99`:

```
# chvg -a n vg99
```

After each IPL, the `syncvg` command must be run manually on the volume group, as in the following example for volume group `vg99`:

```
# syncvg -f -v vg99
```

or specifically for the logical volume `lv99` as follows:

```
# syncvg -f -l lv99
```

However, if the mirroring is only needed to protect the data in the logical volume when the system is up and running, as it is for paging logical volumes (and no manual synchronization after IPL is needed either), MWC should always be turned off, as in the following example for `hd6`:

```
# chlv -w n hd6
```

If mirroring is being used and MWC is on (as it is by default), consider locating the copies in the outer region of the disk, because the MWC information is always written in Cylinder 0 (on the edge of the disk). The following example sets the `lv99` logical volumes interpolicy to edge:

```
# chlv -a e lv99
```

But to actually move the logical volume, reorganization of the disk is necessary, which can be done by the `reorgvg` command, as in the following example for the `vg99` volume group:

```
# reorgvg vg99 lv99
```

or if you want to move the logical volume to another empty disk, use the `migratepv` command, as in the following example:

```
# migraptepv -l lv99 hdisk0 hdisk1
```

Scheduling policy for reading/writing logical partition copies

Different scheduling policies can be set for the logical volume. Different types of scheduling policies are used for logical volumes with multiple copies, as shown in Table 22:

Table 22. Implications on performance of mirror write scheduling policies

Mirror Write Scheduling Policy	Implications
sequential	The sequential policy results in all reads being issued to the primary copy. Writes happen serially, first to the primary disk; only when that is completed is the second write initiated to the secondary disk.
parallel	The parallel policy balances reads between the disks. On each read, the system checks whether the primary is busy. If it is not busy, the read is initiated on the primary. If the primary is busy, the system checks the secondary. If it is not busy, the read is initiated on the secondary. If the secondary is busy, the read is initiated on the copy with the least number of outstanding I/Os. Writes are initiated concurrently.
parallel/sequential	The parallel/sequential policy always initiates reads on the primary copy. Writes are initiated concurrently.
parallel/round robin	The parallel/round robin policy is similar to the parallel policy, except that instead of always checking the primary copy first, it alternates between the copies. This results in equal utilization for reads even when there is never more than one I/O outstanding at a time. Writes are initiated concurrently.

For data that has only one physical copy, the logical volume device driver translates a logical read or write request address into a physical address and calls the appropriate physical device driver to service the request. This single-copy policy handles bad block relocation for write requests and returns all read errors to the calling process.

Tuning scheduling for reading/writing logical partition copies

Mirroring-scheduling policies, such as parallel and parallel/round-robin, can allow performance on read-intensive mirrored configurations to be equivalent to nonmirrored ones. Typically, performance on write-intensive mirrored configurations is not as good than non-mirrored, unless more disks are used.

The default policy usually makes little use of secondary copies when reading, but change the policy to parallel/round-robin, as in the following example for `lv99`:

```
# chlv -d pr lv99
```

Mirroring on different disks

The default for disk mirroring is that the copies should exist on different disks. This is for performance as well as data integrity. With copies residing on different disks, if one disk is extremely busy, then a read request can be completed by the other copy residing on a less busy disk.

Mirroring across different adapters

Another method to improve disk throughput is to mirror the copies across adapters. This will give you a better chance of not only finding a copy on a disk that is least busy, but it will also improve your chances of finding an adapter that is not as busy. LVM does not realize, nor care, that the two disks do not reside on the same adapter. If the copies were on the same adapter, the bottleneck there is still the bottleneck of getting your data through the flow of other data coming from other devices sharing the same adapter card. With multi-adapters, the throughput through the adapter channel should improve unless the adapter bus gets saturated.

Mirroring across different adapters on different busses

Since very high performance adapters, such as the SP Switch 2, SSA or Gigabit Ethernet, can saturate a PCI bus with streaming data by itself, consideration should be made as to the placement of adapters on different busses if more than one bus is available.

6.3.6.2 Write verification

From a performance standpoint, mirroring is costly, mirroring with Write Verify costs even more (extra disk rotation per write), and mirroring with both Write Verify and MWC costs the most (disk rotation plus a seek to Cylinder 0). Write Verify defaults to off, and it does have meaning (and cost) for both mirrored as non-mirrored logical volumes.

When you have write verify enabled, every write to a physical portion of a disk that is part of a logical volume causes the disk device driver to issue the `Write and Verify scsi` command to the disk. This means that after each write, the disk will reread the data and do an IOCC parity check on the data to see if what the platter wrote exactly matched what the write request buffer contained.

6.3.7 Logical volume striping

Striping is a technique for spreading the data in a logical volume across several disk drives in such a way that the I/O capacity of the disk drives can be used in parallel to access data on the logical volume. The primary objective of striping is very high-performance reading and writing of large sequential files, but there are also benefits for random access.

In an ordinary logical volume, the data addresses correspond to the sequence of blocks in the underlying physical partitions. In a striped logical volume, the data addresses follow the sequence of stripe units. A complete stripe consists of one stripe unit on each of the physical devices that contains part of the striped logical volume. The LVM determines which physical blocks on which physical drives correspond to a block being read or written. If more than one drive is involved, the necessary I/O operations are scheduled for all drives simultaneously.

Mirroring and striping at the same time

AIX Version 4.3.3 introduces the new mirroring and striping function (also known as RAID 0+1, Enhanced RAID 0 or RAID 10). This function provides better data availability at the cost of extra disks. In the case of non-mirrored and striped logical volume, failure of one physical volume causes the loss of the whole striped logical volume since there is no redundancy. The mirroring and striping function prevents this situation from happening by adding, up to three mirrors, the striped logical volumes. One failing disk in a striped mirror copy does not bring down the entire logical volume. The remaining (working) disks in the striped mirror copy continue to service striped units. Although the striped units on the failed disk are inaccessible, the mirrored copies of these units are available from another mirror. In this case, users are unaware that a disk is unavailable. To support the mirroring and striping function, a new partition allocation policy (*super strict*) is also introduced. This policy prohibits partitions from one mirror from sharing a disk with a second or third mirror. Volume groups that contain logical volumes that are both striped and mirrored cannot be imported into AIX V4.3.2 or earlier.

LVM writing of data from striped logical volumes

In the write phase, a caller (like the journaled file system device driver) issues one write I/O request to the LVM device driver. This request is chopped into several chunks, and they are written to each physical disk in parallel. Since these writes are achieved on separate drives, they are executed in parallel. It improves the write performance compared to one drive. Subsequently, if all the writes to each physical volume return with no error, then the LVM device driver returns a success to the caller. Otherwise, it returns an error to the caller.

LVM reading of data from striped logical volumes

In the read phase, a caller (like the journal file system device driver) issues one read I/O request to the AIX LVM device driver. This request is split into several read calls for chunks, and these small reads are sent to each physical disk in parallel. Since these reads are achieved on separate drives, they are executed in parallel. It improves the read performance, compared to one physical drive. Subsequently, if all the reads to each physical volume return with no error, the LVM device driver directly passes the memory pointers for all the chunks that compose the data into the user's memory area without re-assembly. If one of the read calls for chunks failed, then the AIX LVM device driver would return an error to the caller.

***Tuning* Logical Volume Striping**

Sequential and random disk I/Os benefit from disk striping. The following techniques have yielded the highest levels of sequential I/O throughput:

- Spread the logical volume across as many physical volumes as possible.
- Use as many adapters as possible for the physical volumes.
- Create a separate volume group for striped logical volumes.
- Set a stripe-unit size of 64 KB.
- Set minpgahead to 2. Example: /usr/samples/kernel/vmtune -r 2.
- Set maxpgahead to 16 times the number of disk drives (16 4 KB pages fills one disks stripe). This causes page-ahead to be done in units of the stripe-unit size (64 KB) times the number of disk drives, resulting in the reading of one stripe unit from each disk drive for each read-ahead operation. For example (with eight disks): /usr/samples/kernel/vmtune -R 128.
- Request I/Os for 64 KB times the number of disk drives. This is equal to the maxpgahead value. For eight disks, it would be 512KB.
- Modify maxfree to accommodate the change in maxpgahead (maxfree = minfree + maxpgahead. For example (with eight disks and default setting of minfree [120]): /usr/samples/kernel/vmtune -F 248.
- Do not mix striped and non-striped logical volumes in the same physical volume. All physical volumes should be the same size within a set of striped logical volumes.

If the striped logical volumes are on raw logical volumes and writes larger than 1.125 MB are being done to these striped raw logical volumes, increasing the lvm_bufcnt parameter with the `vmtune` command might increase throughput of the write activity as shown in the following example (default value is 9):

```
# /usr/samples/kernel/vmtune -u 64
```

The following example shows how to create a RAID 10 logical volume spread over four disks:

```
# mklv -y lv99 -e x -c 2 -s s vg99 4 hdisk1 hdisk2 hdisk4 hdisk5
# lslv -m lv99
lv99:N/A
LP PP1 PV1 PP2 PV2 PP3 PV3
0001 0109 hdisk1 0109 hdisk2
0002 0109 hdisk4 0109 hdisk5
0003 0110 hdisk1 0110 hdisk2
0004 0110 hdisk4 0110 hdisk5
```

6.3.8 JFS and JFS log

The Journaled File System (JFS) uses a database journaling technique to maintain a consistent file system structure. This involves duplicating transactions that are made to file system metadata to the circular JFS log. File system metadata includes the superblock, i-nodes, indirect data pointers, and directories.

6.3.8.1 How to use JFS logs

If possible the JFS log volume should reside on a disk of its own or together with low usage logical volumes if it becomes heavily used (due to application workload). One example of when the log volume will be very active is if hundreds to thousands of SPOOL files are created for the print subsystem per hour.

Having multiple log devices in a volume group is supported. However, a log for a file system must be in the same volume group as that of the file system. A log logical volume or file system logical volume can be moved to another disk using the `migratepv` command, even while the system is running and in use, as in the following example with logical volume lv99 on hdisk1 in the same volume group as hdisk8:

```
# migratepv -l lv99 hdisk1 hdisk8
```

Placing the log logical volume on a physical volume different from your most active file system logical volume will increase parallel resource usage. A separate log for each file system is also supported. To change a log volume for a file system, follow these steps:

1. Create a log volume, as in the following example, that creates the log volume log99 in volume group vg99 consisting of one logical partition:

```
# mklv -y log99 -t jfslog vg99 1
```

2. Format the new log volume:

```
# print y | logform /dev/log99
```

3. Unmount the file system:

```
# umount /filesystem
```

4. Edit the stanza for the file system in the /etc/filesystems file, either manually or by using the following commands:

   a. First, extract all but the /filesystem stanza from /etc/filesystems:

   ```
   # grep -v -p filesystem: /etc/filesystems > /tmp/filesystems.new
   ```

   b. Then, extract only the /filesystem stanza, change the log volume and add it to the new file systems file

   ```
   # grep -p filesystem /etc/filesystems |
      sed 's/hd8/log99/g' >>/tmp/filesystems.new
   ```

   c. Then, make a backup copy of the original /etc/filesystems file:

   ```
   cp /etc/filesystems /etc/filesystems~1
   ```

   d. Finally, copy the new file systems file to the /etc directory:

   ```
   cp /tmp/filesystems.new /etc/filesystems
   ```

5. Run `fsck` on the file system (not always necessary):

```
fsck -y /filesystem
```

6. Mount the file system again:

```
mount /filesystem
```

The only drawback to the preceding procedure is that the new file system stanza will be added to the end of the /etc/filesystems file and thus mounted last. The following script can be used to fix it right and put the modified stanza back where it was:

```
#!/bin/ksh

fs=${1:-home}
oldlog=${2:-hd8}
newlog=${3:-log99}
tmpfile=/tmp/filesystems.new

integer n=$(grep -n -p $fs: /etc/filesystems | sed 's/://g;1p;1,$d')
((n=$n-3))
grep -v -p $fs: /etc/filesystems > $tmpfile
cat <<-! | ed $tmpfile
    ${n}r ! grep -p $fs: /etc/filesystems | sed 's/hd8/log99/'
    w
!
cp /etc/filesystems /etc/filesystems~1
cp $tmpfile /etc/filesystems
rm -f $tmpfile
```

6.3.8.2 How to not use the JFS log

The `mount` option nointegrity bypasses the use of a JFS log for the file system mounted with this option. This can provide better performance as long as the administrator knows that the `fsck` command might have to be run on the file system if the system goes down without a clean shutdown.

```
# mount -o nointegrity /filesystem
```

To make the change permanent, use the `chfs` command as follows:

```
# chfs -a options=nointegrity,rw /filesystem
```

6.3.9 Paging space allocation

If paging space is needed in a system, performance and throughput will always suffer; thus, the obvious conclusion is to eliminate paging to paging space as much as possible.

The current default paging-space-slot-allocation method, *Deferred Page Space Allocation* (DPSA), delays allocation of paging space until it is necessary to page out the page.

At some point in time, allocated paging space might be needed; to avoid problems arising from having too little virtual memory, creation of sufficient space is still necessary. There are a couple of schools of thought concerning the creation of paging space:

- Provide a Dedicated disk for a single paging spaced.
- Use single paging space spread over multiple shared disks.
- Have multiple paging spaces allocated, one each on multiple shared disks.

However, what is important when creating paging spaces is that the disk(s) are the least used ones. When paging is needed, it should not have to compete with other disk operations; keep in mind that if paging to paging space occurs, and processes are delayed because of it, performance will suffer.

Apart from optimizing the speed of accessing paging space, as for any logical volume, there are a few extra considerations to take, due to the fact that paging spaces are not like ordinary logical volumes. Paging spaces are accessed in a round robin fashion and the data stored in the logical volume are of no interest after a reboot (IPL). Therefore, disable mirror write consistency (MWC) for paging spaces that are mirrored. Do not allocate more than one paging space per physical disk, and use the *center* intra-disk allocation policy (except when the paging space coexists on the same disk as mirrored logical volumes; then it should have the *edge* intra-disk policy). Do not let paging spaces and JFS log volumes share the same disk, and if possible, let the paging spaces reside on separate disk(s).

The following example shows the creation of a striped paging space:

```
# mklv -y hd66 -t paging -a c -S 64K rootvg 110 hdisk2 hdisk3 hdisk4 hdisk5
hd66
```

After creating the paging space, we need to activate it:

```
# chps -a y hd66 && swapon /dev/hd66
```

This is the output from the `lsps` command:

```
# lsps -a
Page Space  Physical Volume   Volume Group    Size   %Used  Active  Auto  Type
hd66        hdisk2            rootvg         896MB       1     yes   yes    lv
hd66        hdisk3            rootvg         896MB       1     yes   yes    lv
hd66        hdisk4            rootvg         896MB       1     yes   yes    lv
hd66        hdisk5            rootvg         896MB       1     yes   yes    lv
hd6         hdisk0            rootvg        9216MB       1     yes   yes    lv
```

6.3.10 Tuning I/O pacing

Disk I/O pacing is intended to prevent programs that generate very large amounts of output from saturating the system's I/O facilities and causing the response times of less-demanding programs to deteriorate. Disk I/O pacing enforces per-segment (which effectively means per-file) high- and low-water marks on the sum of all pending I/Os. When a process tries to write to a file that already has high-water mark pending writes, the process is put to sleep until enough I/Os have completed to make the number of pending writes less than or equal to the low-water mark. The logic of I/O request handling does not change. The output from high-volume processes is slowed down somewhat.

One limitation of pacing is that it does not offer as much control when a process writes buffers larger than 4 KB. When a write is sent to the VMM and the high-water mark has not been met, the VMM performs start I/Os on all pages in the buffer, even if that results in exceeding the high-water mark. Pacing works well on the `cp` command, because the `cp` command writes 4 KB at a time.

Disk-I/O pacing is a tuning parameter that can improve interactive response time in some situations where foreground or background programs that write large volumes of data are interfering with foreground requests. If not used properly, however, it can massively reduce throughput.

Note

Before using I/O pacing restrictions, analyze the disk and adapter utilization and try to move disks or file systems to even out the I/O load. If this is not an option, or has already been done and there still are processes that hog the I/O bandwidth, use I/O pacing and check if the overall throughput will increase over a given time period.

Programs whose presence in a workload that could benefit from disk-I/O pacing include:

- Programs that generate large amounts of output algorithmically, and thus are not constrained by the time required to read input. Some such programs may need pacing on comparatively fast processors and not need it on comparatively slow processors.
- Programs that write large, possibly somewhat modified files that have been read in their entirety shortly before writing begins (by a previous command, for example).
- Filters, such as the `tar` command, that read a file and write it out again with little processing. The need for pacing can be exacerbated if the input is being read from a faster disk drive than the output is being written to.

The pseudo device driver sys0 parameters maxpout and minpout can be used to enable and configure I/O pacing:

maxpout — Sets the maximum number of I/O write requests that are allowed against a single file.

minpout — Sets the number of I/O requests against a single file before the issuing processes are resumed.

The high- and low-water marks are not straightforward because of the combination of write-behind and asynchronous writes. Start by using the following simple formula to calculate the maxpout setting (where N is a positive integer such as 4, 8, 16):

maxpout = (4 * N) + 1

minpout = (maxpout - 1) / 2

The settings in Table 23 are a good place to start, but some experimenting will be needed to find the best settings for your workload. Remember that the aim with I/O pacing is to increase overall I/O throughput during a given time period.

Table 23. Sample values for minpout and maxpout

N	minpout	maxpout
4	8	17
8	16	33
16	32	65

The following examples set the maximum number of pending I/O write requests outstanding against a file to 33 (maxpout) before suspending the process.

```
# chdev -l sys0 -a maxpout=33 -a minpout=16
```

Once the process is suspended, it will not be resumed until the outstanding I/O write requests against the file drop to 16 (minpout).

Note

Setting minpout and maxpout to 0 disables I/O pacing (the default setting is 0 for both).

6.3.11 Asynchronous I/O

Synchronous I/O occurs while you wait. Applications processing can not continue until the I/O operation is complete. In contrast, asynchronous I/O operations run in the background and do not block user applications. This improves performance, because I/O operations and applications processing can run simultaneously.

Using asynchronous I/O will usually improve I/O throughput. For JFS I/O, the actual performance, however, depends on how many server processes are running that will handle the I/O requests.

Note

In AIX V4, async I/O on JFS file systems is handled by kprocs. Asynchronous I/O on raw logical volume partitions is handled directly by the kernel.

Many applications, such as databases and file servers, take advantage of the ability to overlap processing and I/O. These asynchronous I/O operations use various kinds of devices and files. Additionally, multiple asynchronous I/O operations may run at the same time on one or more devices or files.

Applications use the aio_read() and aio_write() subroutines to perform the I/O. Control returns to the application from the subroutine as soon as the request has been queued. The application can then continue processing while the disk operation is being performed.

A kernel process (KPROC), called a server, is in charge of each request from the time it is taken off the queue until it completes (for JFS). The number of servers limits the number of JFS disk I/O operations that can be in progress in the system simultaneously.

> **Note**
>
> AIO will not relieve an overly busy disk drive. Using the `iostat` command with an interval and count value, you can determine if any disks are overly busy. Monitor the %tm_act column for each disk drive on the system. On some systems, a %tm_act of 35.0 or higher for one disk can cause noticeably slower performance. The relief for this case could be to move data from more busy to less busy disks, but simply having AIO will not relieve an overly busy disk problem.

To activate asynchronous I/O during boot (IPL), run this command:

```
# chdev -l aio0 -a autoconfig=available
```

To activate asynchronous I/O in the running system without doing a reboot, run this command:

```
# mkdev -l aio0
```

The following command will tell you how many AIO Servers are currently running (you must run this command as the *root* user):

```
# pstat -a | grep -c aios
```

If the disk drives that are being accessed asynchronously are using the Journaled File System (JFS), all I/O will be routed through the aios KPROCs. If the disk drives that are being accessed asynchronously are using a form of RAW logical volume management, then the disk I/O is not routed through the aios KPROCs. In that case, the number of servers running is not relevant. To confirm that an application that uses RAW logical volumes are taking advantage of asynchronously I/O, you can disable the fast path option for the device driver (thus forcing all I/O activity through the aios KPROCs) using the following command:

```
# chdev -l aio0 -a fastpath=disable
```

When this option has been disabled, even RAW I/O will be forced through the aios KPROCs. At that point, the `pstat` command listed above will work for RAW I/O applications as well. You would not run the system with this option disabled for any length of time. This is simply a suggestion to confirm that the application is working with asynchronously I/O and RAW logical volumes.

Functions provided by the asynchronous I/O facilities are:

- Large File-Enabled Asynchronous I/O
- Nonblocking I/O
- Notification of I/O completion
- Cancellation of I/O requests

One fundamental limitation in asynchronous I/O is page hiding. When an unbuffered (raw) asynchronous I/O is issued, the page that contains the user buffer is hidden during the actual I/O operation. This ensures cache consistency. However, the application may access the memory locations that fall within the same page as the user buffer. This may cause the application to block, as a result of a page fault. To alleviate this situation, the application should allocate page aligned buffers and not touch the buffers until the I/O request using it has completed.

6.3.11.1 Tuning asynchronous I/O

The default values are 1 for minservers and 10 for maxservers. In systems that seldom run applications that use asynchronous I/O, this is usually adequate. For environments with many disk drives and key applications that use asynchronous I/O, the default is far too low. The result of a deficiency of servers is that disk I/O seems much slower than it should be. Not only do requests spend inordinate lengths of time in the queue, but the low ratio of servers to disk drives means that the seek-optimization algorithms have too few requests to work with for each drive.

Using the `vmstat` command with an interval and count value, you can determine if the CPU is idle (waiting for disk I/O). The `wa` column details the percentage of time the CPU was idle with pending local disk I/O. If there is at least one outstanding I/O to a local disk when the wait process is running, the time is classified as waiting for I/O. Unless asynchronous I/O is being used by the process, an I/O request to disk causes the calling process to block (or sleep) until the request has been completed. Once a process's I/O request completes, it is placed on the run queue.

The asynchronous I/O device has a few attributes that regulate it’s performance impact:

maxreqs — The maxreqs attribute is the maximum number of asynchronous I/O requests that can be outstanding at one time. This includes requests that are in progress as well as those that are waiting to be started. The maximum number of asynchronous I/O requests cannot be less than 4096 but it can be greater. It would be appropriate for a system with a high volume of asynchronous I/O to have a maximum number of asynchronous I/O requests larger than 4096.

kprocprio The kprocprio attribute is the priority level of kernel processes dedicated to asynchronous I/O. The lower the priority number is, the more favored the process is in scheduling. Concurrency is enhanced by making this number slightly less than the value of 40, the priority of a normal user process. It cannot be made lower than the value of 16.

Since the default priority is (40+nice), these daemons will be slightly favored with this value of (39+nice). If you want to favor them more, make changes slowly. A very low priority can interfere with other system processes.

Important

Raising kprocprio (decreasing this numeric value) is not recommended because system hangs or crashes could occur if the priority of the asynchronous I/O servers are favored too much. There is little to be gained by making big priority changes.

minservers The minservers attribute are the minimum number of kernel processes dedicated to asynchronous I/O processing. Since each kernel process uses memory, this number should not be large when the amount of asynchronous I/O expected is small.

maxservers The maxservers attribute are the maximum number of kernel processes dedicated to asynchronous I/O processing. There can never be more than this many asynchronous I/O requests in progress at one time, so this number limits the possible I/O concurrency.

Setting minservers and maxservers

If the number of asynchronous I/O requests is high, then the basic recommendation is to increase maxservers to the approximate number of possible simultaneous I/Os. In most cases, it is better to leave the minservers parameter at the default value, because the asynchronous I/O kernel extension will generate additional servers (if needed).

Here are some suggested rules of thumb for determining what value to set the maximum number of servers to:

1. The first rule of thumb suggests that you limit the maxservers to a number equal to ten times the number of disks that are to be used concurrently, but not more than 80. The minservers should be set to half of maxservers.
2. Another rule of thumb is to set maxservers to 80 and leave minservers set to the default of 1 and reboot. Monitor the number of additional servers started throughout the course of normal workload. After a 24-hour period of normal activity, set maxservers to (The number of currently running aios + 10), and set minservers to (The number of currently running aios - 10).
3. For environments with more than 80 aio KPROCs running, take statistics using `vmstat -s` before any high I/O activity begins and again at the end. Check the field iodone. From this field, you can determine how many physical I/Os are being handled in a given wall clock period. Then increase maxservers and see if you can get more iodones in the same time period.

6.3.12 Logical volume policies

The *inter-disk* and *intra-disk* allocation policies can have performance impact and needs to be evaluated regarding how applications use data, and how much data is read and written.

6.3.12.1 Inter-disk allocation policy

The inter-disk allocation policy specifies the number of disks on which a logical volume's physical partitions are located. The physical partitions for a logical volume might be located on a single disk or spread across all the disks in a volume group.

Minimum	The minimum setting indicates that one physical volume should contain all the original physical partitions of this logical volume if possible. If the allocation program must use two or more physical volumes, it uses the minimum number, while remaining consistent with other parameters.
Maximum	The maximum setting, considering other constraints, spreads the physical partitions of the logical volume as evenly as possible over as many physical volumes as possible. This is a performance-oriented option, because spreading the physical partitions over several disks tends to decrease the average access time for the logical volume. To improve availability, the maximum setting should only be used with mirrored logical volumes.

Tuning considerations

When choosing between minimum or maximum inter-disk allocation policy, other factors should be taken more into account than just the individual logical volume. Questions such as these must be asked and evaluated:

1. Will other heavily used logical volumes occupy the same disk?
2. Will the reading and writing be done in a sequential or random way?
3. How much data will be read or written with each I/O operation?
4. How much of a load on the logical volume will saturate the I/O channel, from memory to disk, if one adapter and minimum inter-disk allocation policy is used and no other applications are contending for throughput?

6.3.12.2 Intra-disk allocation policy

The intra-disk allocation policy choices are based on the five regions of a disk where physical partitions can be located. The five regions are: *outer edge*, *inner edge*, *outer middle*, *inner middle*, and *center*. The edge partitions have the slowest average seek times, which generally result in longer response times for any application that uses them. The center partitions have the fastest average seek times, which generally result in the best response time for any application that uses them. There are, however, fewer partitions on a physical volume at the center than at the other regions.

The JFS log is a good candidate for allocation at the center of a physical volume because it is used by the operating system so often. At the other extreme, the boot logical volume is used infrequently and should be allocated at the edge or middle of the physical volume.

The general rule is that the more I/Os, either absolutely or during the running of an important application, the closer to the center of the physical volumes the physical partitions of the logical volume should be allocated. This rule has two important exceptions:

- Logical volumes that contain large, sequential files could benefit from being located at the edge, if the disks in question have more blocks per track at the edge than farther in (resulting in better sequential performance).
- Mirrored logical volumes with Mirror Write Consistency (MWC) set to ON should be at the outer edge, because that is where the system writes MWC data.

Note

When creating large quantities of logical volumes, consider the consequences of selecting the center allocation policy.

When creating large logical volumes that span multiple disks, start by using the *edge* inter-disk allocation policy and let AIX create the logical volumes consecutively, starting from the edge. If allocated from the center, there is a risk that the logical volumes, after the first one, will be split around the previous logical volume(s) that occupy most of the center already.

6.3.12.3 Logical volume reorganization

In our following example, we have a file system that is heavily used. We will spread the logical volume over as many disks as possible in the volume group, but away from disks that are heavily used.

First, we change the intra-policy for the logical volume to maximum (and while we are at it, we set the intra-policy to center and the scheduling policy to parallel round robin, in case we will turn on mirroring later) by using the following command:

```
# chlv -a c -e x -d pr lv01
```

We check the following output to see the logical volume layout in the volume group:

```
lslv -l lv01
lv01:/testfs2
PV              COPIES         IN BAND       DISTRIBUTION
hdisk1          032:000:000    0%            000:028:000:004:000
```

So, because we have not done so before, we extend the volume group with a couple of disks, use the following command:

```
# extendvg vg99 hdisk2 hdisk3 hdisk4 hdisk5
```

We can now reorganize the logical volume so we will get equivalent number of partitions on each disk. To do this, we use `migratepv`:

```
# migratepv -l lv01 hdisk1 hdisk2 hdisk3 hdisk4 hdisk5
```

We now check how the logical volume layout in the volume group has changed:

```
lslv -l lv01
lv01:/testfs2
PV              COPIES          IN BAND       DISTRIBUTION
hdisk2          012:000:000     100%          000:000:012:000:000
hdisk3          008:000:000     100%          000:000:008:000:000
hdisk4          006:000:000     100%          000:000:006:000:000
hdisk5          006:000:000     100%          000:000:006:000:000
```

6.3.12.4 Volume group fragmentation

If file systems (logical volumes) have been expanded after the first creation, it is very likely that the logical volume will have physical partitions in a non-sequential manner and might even spread over several disks in the volume group in this way. To consolidate all space needed for a logical volume in a consecutive way, the `reorgvg` command can be used. This command is used to move the placement of logical volumes in physical volumes by using lower level commands that mirrors the physical partitions, syncs them and then removes the unwanted copy. The following example illustrates the command usage for the *spdatavg* volume group:

```
# reorgvg spdatavg
0516-962 reorgvg: Logical volume loglv01 migrated.
0516-962 reorgvg: Logical volume lv06 migrated.
```

6.3.12.5 Example scenario

As an example, we will run a RDBMS server that will read and write 8 KB each time from database files residing in JFS file systems. We have 16 SSA disks, and two SSA adapters, and we will use mirroring for availability. Since the RDBMS will mostly read randomly (the application is heavily indexed and very few full table scans will occur), we will not use striping. We will also do a standard split of the SSA disks between the two adapters and use two loops.

Since the main usage will be for the RDBMS, we only have to decide the optimal placing of the logical volumes for the RDBMS. Since there is an identical system running on a different location, we can determine the I/O requirements by monitoring the RDBMS and AIX. Since our investigation of the running system concludes that there are 14 tables, of which 1/3 are currently heavily used, 1/3 is rarely used, and 1/3 are in between, we decide to use *maximum* inter-disk allocation policy and create almost one logical volume for each table. Since we only use the SSA drawer for the RDBMS, we do not have to worry about the JFS log for the application, because the application will not write outside the preallocated database files. The most used 1/3 logical volumes were also allocated as close to the *edge* part on the disks, the medium used 1/3 in the *middle*, and the least used 1/3 at *center*. After production starts on the system, we see an evenly balanced I/O situation over most disks and the throughput is perceived to be good by the end-users.

However, since a lot more SPOOL files are created during production than was originally anticipated (on other disks), we created a file system for SPOOL files on the SSA disk volume group. We put this file system on the first disk together with the JFS log, and because we allocated most of the available space on the disks already, we change the policy from *maximum* to *minimum* for the rarely used tables. And then we use `migratepv` to move them from the other disks to the first one. Sometimes `reorgvg` will work, but when we consolidate from a lot of space-constrained disks to one specific disk, `migratepv` gives the desired level of control and manageability. The JFS log was moved to the center of the disk, as was the new SPOOL file system, and MWC was turned off for both the SPOOL file system and the JFS log.

The above example illustrates that different policies can be utilized for different logical volumes that are used by the same application because there are always trade-offs to be made in a production environment.

6.3.13 File system fragmentation

When a file system is created, the system administrator can specify the size of the fragments in the file system. The allowable sizes are 512, 1024, 2048, and 4096 bytes (the default). Files smaller than a fragment are stored in a single fragment, conserving disk space, which is the primary objective.

Files smaller than 4096 bytes are stored in the minimum necessary number of contiguous fragments. Files whose size is between 4096 bytes and 32 KB (inclusive) are stored in one or more (4 KB) full blocks and in as many fragments as are required to hold the remainder. For example, a 5632-byte file would be allocated one 4 KB block, pointed to by the first pointer in the i-node. If the fragment size is 512, then eight fragments would be used for the first 4 KB block. The last 1.5 KB would use three fragments, pointed to by the second pointer in the i-node. For files greater than 32 KB, allocation is done in 4 KB blocks, and the i-node pointers point to these 4 KB blocks.

Whatever the fragment size, a full block is considered to be 4096 bytes. In a file system with a fragment size less than 4096 bytes, however, a need for a full block can be satisfied by any contiguous sequence of fragments totalling 4096 bytes. It need not begin on a multiple of 4096 byte boundary.

The file system tries to allocate space for files in contiguous fragments by spreading the files themselves across the logical volume to minimize interfile allocation interference and fragmentation.

The primary performance hazard for file systems with small fragment sizes is space fragmentation. The existence of small files scattered across the logical volume can make it impossible to allocate contiguous or closely spaced blocks for a large file. Performance can suffer when accessing large files. Carried to an extreme, space fragmentation can make it impossible to allocate space for a file, even though there are many individual free fragments.

Another adverse effect on disk I/O activity is the number of I/O operations. For a file with a size of 4 KB stored in a single fragment of 4 KB, only one disk I/O operation would be required to either read or write the file. If the choice of the fragment size was 512 bytes, eight fragments would be allocated to this file, and for a read or write to complete, several additional disk I/O operations (disk seeks, data transfers, and allocation activity) would be required. Therefore, for file systems which use a fragment size of 4 KB, the number of disk I/O operations might be far less than for file systems which employ a smaller fragment size.

6.3.13.1 How to defragment a file system

To check if a file system is fragmented and would benefit from defragmentation (a process which on large partitions will take considerable time and resource), use the `defragfs` command, as in the following example:

```
# defragfs -q /filesystem
statistics before running defragfs:
number of free fragments 239445
number of allocated fragments 391339
number of free spaces shorter than a block 0
number of free fragments in short free spaces 0
```

Figure 21. Querying fragmentation for a file system

In the output from the `defragfs` command, check the following values:

- Number of free spaces shorter than a block
- Number of free fragments

If the number of free spaces shorter than a block is high or close to the number of free fragments, use `defragfs` to consolidate the free space. The following example shows how the fragments for files in the specified file system are consolidated:

```
# defragfs /filesystem
statistics before running defragfs:
number of free fragments 239445
number of allocated fragments 391339
number of free spaces shorter than a block 0
number of free fragments in short free spaces 0

statistics after running defragfs:
number of free spaces shorter than a block 0
number of free fragments in short free spaces 0

other statistics:
number of fragments moved 2713
number of logical blocks moved 2713
number of allocation attempts 937
number of exact matches 129
```

Figure 22. Defragmentation of file system with defragfs

6.3.13.2 How to rearrange datablocks in a file system

To rearrange all data blocks so that files and directories have their respective data blocks arranged contiguously in the file system, the file system has to be rebuilt. There is no software available from IBM[8] to perform this task as `defragfs` does for fragments. To achieve the same result as `defragfs` the following must be done:

1. Backup the file system data (use `tar`, `cpio`, `backup` or a similar command) and verify that it is restorable before continuing.
2. Unmount the file system (use the `umount /filesystem`).
3. Rebuild the file system (use the `print yes|mkfs /filesystem`).
4. Mount the file system again (use the `mount /filesystem`).
5. Restore the backup of the file system data.

8 Eagle Software (www.eaglesoft.com) has software for this.

Chapter 7. Control workstation tuning

This chapter discusses various control workstation (CWS) considerations and the tuning of its performance.

7.1 Control workstation considerations

The CWS serves as a single point of control for various PSSP subsystems that provide configuration data, security, hardware monitoring, diagnostics, and, optionally, job scheduling data and a time source for the SP nodes.

It is important that there are necessary resources available when these PSSP subsystem requires them. Thus, it is recommended not to use the CWS to run other programs or applications.

Do not use the CWS as a server for any other applications, for example, TSM or Netview, because they can deprive necessary resources from CWS and interfere with PSSP subsystems. If this happens, it may cause an unexpected interruption to the operation of your production SP nodes.

One probable exception to this is the Performance Toolbox. If we do not have other machines that can run it, we can run it on CWS and display the consoles on the local display. (We do not recommend displaying the consoles on the remote machine, because this can place quite a workload on the CWS.)

Do not use the CWS as a client for other applications. If this can not be avoided, be aware of the impact that the client function can cause to the overall performance of the CWS and try to prevent that from happening. For example, if the CWS is a client of a DNS server and the server appears to be down, the DNS client on CWS will retry to contact the name server several times until it times out. This can have serious performance impact to the CWS. In this case, we recommend that you put all hostnames required by the CWS in the /etc/hosts file and create a /etc/netsvc.conf file with a line saying: “host=local,bind”, as this instructs AIX to consult local /etc/hosts file before going to the DNS server for name resolution.

In summary, dedicate the CWS to the PSSP management and monitoring tasks, such as Perspectives and Performance Toolbox, only.

7.2 Control workstation tuning

This section describes various tuning parameters for the CWS.

7.2.1 Network tunables

When you first install AIX on the CWS, its network tunable values are set to the default. Your system may not run efficiently with these values.

When installing PSSP on your CWS, modify the following network tunables to the values suggested in Table 24.

Table 24. CWS network tunables

Tunable	Recommended initial value
thewall	Leave it at the default value
sb_max	163840
ipforwarding	1
tcp_sendspace	65536
tcp_recvspace	65536
udp_sendspace	32768
upd_recvspace	32768
tcp_mssdflt	1448

When you use the `no -o` command to change a network tunable value, it takes effect immediately. However, it is not preserved upon reboot.

To make the changes to the network tunables effective upon reboot, add the `no -o` command to the bottom of the /etc/rc.net file.

Notice that this is different from tuning the nodes in the SP. The dynamic tuning changes for SP nodes should be done in the /tftpboot/tuning.cust file.

Refer to Chapter 4, “Network tuning” on page 21 for more detailed information about these network tunables.

Note

Because the CWS can perform various roles, for example, to install the nodes, to control/monitor the nodes and so on through the life cycle of an SP system, there may be cases where the initial network tunable options described here may not be optimum.

In that case, use the recommendations in Section 4.7.4, "Tuning for server environments" on page 91.

7.2.2 Adapter tuning

Various models of the network adapters can have different values for transmit and receive queue sizes. These values sometimes also differ depending on the level of AIX you are using.

You can set these values using SMIT or the `chdev` command. We recommend you use SMIT, because PF4 in SMIT shows you the range of valid values that can be specified.

We recommend to set the transmit and receive queue sizes to the maximum value.

If the adapter you are changing is also the adapter for the network you are logged in through or it is being used, you will have to make the changes to the databases only. Then reboot the CWS for the changes to become effective.

Refer to Section Chapter 5., "Adapter tuning" on page 95 for more detail information about network adapter tuning.

7.2.3 Other AIX components tuning

For tuning other AIX components (memory, disk and so on) on the CWS, refer to Chapter 6, "Tuning other components in AIX" on page 115.

7.2.4 Other considerations

The following are other considerations that may have impact on the CWS.

SP Ethernet tuning

Refer to Section 5.3, "SP Ethernet tuning" on page 110 for information about SP ethernet tuning.

Tuning the Topology Services subsystem

Topology Services is a distributed subsystem of the IBM RS/6000 Cluster Technology (RSCT) software for the RS/6000 system. It provides other high availability subsystems with network adapter status, node connectivity information and a reliable message service.

Topology Services is meant to be sensitive to the adapter and network events in the system; this sensitivity is tunable. However, some conditions can degrade the ability of Topology Services to accurately report on adapter or node membership.

One such condition is the failure of AIX to schedule the daemon process in a timely manner. This can cause daemons to be late in sending their heartbeats by a significant amount. This can happen because an interrupt rate is too high, the rate of paging activity is too high, or there are other problems.

If the daemon is prevented from running for enough time, the node might be considered to be down by other peer daemons. The *node down* indication, when propagated to subsystems like VSD and GPFS, will cause these subsystems to perform; in this case, undesirable recovery procedures that take over resources and roles of the node will occur.

Two parameters in the TS_Config class in SDR controls this operation:

Frequency Controls how often Topology Services sends a heartbeat to its neighbors. The value is interpreted as the number of seconds between heartbeat. The minimum and default value is 1 second.

Sensitivity Controls the number of missed heartbeat messages that will cause a *Death in Family message* to be generated. The default is 4. Heartbeats are not considered missing until it has been twice the interval indicated by the frequency value.

The default settings of these values are overly aggressive for a SP system partition with more than 128 nodes or under heavy load conditions. Using the default setting in these environment can result in *false failure indications*.

Decide which settings are suitable for your system by considering the following:

- Higher values for the frequency attributes result in lower CPU and network utilization from the Topology Services daemon.
- Higher values for the product of frequency times sensitivity result in less sensitivity of Topology Services to factors that cause the daemon to be blocked or messages not to reach their destinations. Higher values also

result in Topology Services taking longer to detect a failed adapter or node.

- If the nodes are used primarily for parallel jobs, use the settings shown in Table 25:

Table 25. Settings for nodes running primarily parallel jobs

Frequency	Sensitivity	Seconds to detect failure
2	6	24
3	5	30
3	10	60
4	9	72

- If the nodes are used primarily for database workloads or a mixed environment, use the settings shown in Table 26:

Table 26. Settings for nodes running primarily DB or mixed loads

Frequency	Sensitivity	Seconds to detect failure
2	6	24
3	5	30
2	10	40

- If the nodes tends to operate in a heavy paging or I/O intensive environment, as is often the case when running the GPFS software, use the settings shown in Table 27:

Table 27. Settings for nodes with intensive I/O and heavy paging

Frequency	Sensitivity	Seconds to detect failure
1	12	24
1	15	30

These tunables are typically set after PSSP is installed and configured on the CWS and before installing the nodes. After PSSP has been installed and configured on the nodes, you must refresh the subsystem after making any tuning adjustments.

To display the current settings, use the following command:

```
root@sp6en0:/: /usr/sbin/rsct/bin/hatstune -v
Current HATS tunable values:
    Frequency: 1
    Sensitivity: 6
    Running fixed priority: 38
    Maximum number of lines in log file: 5000
    Pinning in real memory:
```

To change the frequency to 3 seconds and sensitivity to 5, use the following command:

```
root@sp6en0:/: /usr/sbin/rsct/bin/hatstune -f 3 -s 5 -r
The following HATS tunables are changed:
    Default heart beat frequency changed to 3.
    Default heart beat sensitivity changed to 5.
0513-095 The request for subsystem refresh was completed successfully.
```

The -f flag sets the frequency, the -s flag sets the sensitivity and the -r flag refreshes the hats subsystem after the setting.

Note

Topology Services sets the following network options to 1 so that the reliable message feature, which utilizes IP source routing, will continue to work:

- nonlocsrcroute
- ipsrcroutesend
- ipsrcrouterecv
- ipsrcrouteforward

Disabling any of these network options can prevent the reliable message from working properly

Part 3. Problem determination and tools

Chapter 8. Problem determination

This chapter provides you with an initial *Performance Problem Checklist* when your system is experiencing performance problems.

For more information on network tunables, ARP, or NFS tuning, you can refer to Chapter 4, “Network tuning” on page 21. Transmit queue overflows and switch send pools are discussed in Chapter 5, “Adapter tuning” on page 95. Resource monitors, such as `vmstat`, `iostat`, and `netstat`, are explained in more detail in Chapter 10, “Resource monitoring” on page 203.

The `dsh` commands given below are to be run on the nodes. The quoted part of these commands also need to be run on the control workstation.

8.1 Performance problem checklist

Users - Check if the expected users are logged onto the system:

```
dsh -a “who”
```

Processes - Check for runaway processes:

```
dsh -a “ps -gvc”
```

- Look for processes causing lots of pagein counts.
- Look for processes using up lots of memory.
- Look for processes using lots of CPU.

(kproc is the cpu wait process and can be ignored)

Paging - To check the total amount of paging being used:

```
dsh -a “lsps -a”
```

use `vmstat 2 5` to check paging rates:

- Check for page scan rates in the "sr" column.
- Check for page freeing rates in the "fr" column.

CPU - Check CPU utilization:

```
dsh -a “vmstat 2 5”
```

- Check for the percentage of user time in “us” column.
- Check for the percentage of system time in “sys” column.

I/O - Check I/O utilization:

```
dsh -a "iostat 2 5"
```

- Check for imbalance of workload between disks.
- Check %iowait for the time the CPU was idle while there was an outstanding I/O request.

AIX network tunables - Check AIX network tunables:

```
dsh -a "no -a"
```

Transmit queue overflows -Check network adapters for transmit queue overflow errors, particularly on the adapters to the SP Ethernet:

```
dsh -a "netstat -v en0 | grep Overflow "
```

For the switch:

```
dsh -a "estat -d css0 | grep Overflow"
```

ARP cache - Check to see that the ARP cache is set correctly for systems greater than 128 nodes:

```
dsh -a "arp -a | wc -l"
```

Switch send pool (SP Switch only) **-** Check vdidl3 for failed counts on the switch send pool:

```
dsh -a "/usr/lpp/ssp/css/vdidl3 -i"
```

Traffic flow - Check for unexpected or unbalanced traffic flow:

```
dsh -a "netstat -i"
```

NFS - Check for client timeout and retry counts:

```
dsh -a "nfsstat -c"
```

Chapter 9. Common performance problems

When tuning an SP system, some problems can appear that cause performance that is lower than expected. The following sections shed more light on most of these problems, how to detect them, and what tuning parameters to change in order to alleviate them.

9.1 The Nagle Algorithm

A problem that often occurs on the SP systems is that an application runs very slowly when using the SP Switch, while performance is significantly better on an Ethernet or FDDI network interface. This problem is sometimes caused by the Nagle Algorithm (used by TCP/IP) interacting with the delayed ACK (acknowledgement) timer. Complete information about the Nagle Algorithm can be found in Section 4.2.9, “Description of the Nagle Algorithm” on page 42.

The following are suggestions on how to avoid the Nagle Algorithm in SP systems:

- If you are running an application and do not have access to the source code, use the `no` command to increase the TCP window.

  Be careful with this because it may not always be effective. Increasing the tcp_sendspace and tcp_recvspace sizes on the sending and receiving nodes may cause other negative effects on the SP system or to other applications running on the system. Make sure that you set rfc1323 to 1 if the window size exceeds 65536.

- Change the MTU size of the switch.

  Changing the MTU of the switch moves the window and buffer size combination where the 200 msec delay is invoked. When writing 32 KB buffers to a TCP connection, if the TCP/IP window is 65536, only 5 packets/second are transmitted. If you change the MTU of the switch interface to 32768, there is no delay on transmitting a 32768 buffer, because it is the same size as the segment size of the switch. However, reducing the MTU of the switch to 32768 degrades the peak TCP/IP throughput slightly. Reducing the MTU even further degrades the peak throughput even more.

- From within an application, you can increase the TCP window size on the socket by using the SO_SNDBUF and SO_RCVBUF settings on the socket.

- For good performance across a switch, we suggest that both SO_SNDBUF and SO_RCVBUF be set to at least 524288 on both the client and server nodes. You need to set both sides, because TCP uses the lowest common size to determine the actual TCP window.
- If you set the SO_SNDBUF and SO_RCVBUF sizes larger than 65536, you need to set TCP_RFC1323 also on the socket unless the `no` options already set it. Setting TCP_RFC1323 to 1 takes advantage of window sizes greater than 65536.
- You also want to ensure that the system setting for sb_max is at least twice the TCP window size, or sb_max will reduce the effective values for SO_SNDBUF and SO_RCVBUF.

• Set TCP_NODELAY on the socket of the sending side to turn off the Nagle Algorithm.

All data sent will go immediately, no matter what the data size. However, if the application sends very small buffers, you will significantly increase the total number of packets on the network.

Note

Two common problems that can cause unexpected Nagle behavior:

• Set tcp_sendspace and tcp_recvspace to a higher number and forget to set rfc1323 to 1 on all nodes.

• Set tcp_sendspace and tcp_recvspace large on one node and not in the other.

On systems where a node is talking to several other nodes, it is harder to see the Nagle effect. In this case, the only way to detect it is to examine the `iptraces` command outputs to extract a single socket's traffic. What can happen is that if one node is talking to two nodes, each connection can have the Nagle effect, and the packet rate over the switch is 10 packets/sec. If you have one node talking to five nodes, the packet rate can be 25 packets/second, but the aggregate switch traffic 1.5 MB/sec. This rate exceeds the throughput on a slow Ethernet network, but is well below what the switch can handle.

9.2 Adapter queue overflows

Many types of network adapters are supported in the RS/6000 SP environment. When data is being sent, it is passed through the TCP/IP layers to the device driver for that adapter. At the device driver layer, data is queue in network memory pool space until it is transmitted by the adapter. Then, it is signaled to start DMA operations. When the adapter has completed the transmission, it interrupts the system, and the device interrupt routines are called to adjust the transmit queues and free the network memory buffers that held the transmitted data.

When frames are received by the adapter, they are transferred from the adapter into a driver-managed receive queue. This queue consists of network memory buffers. If no network memory buffers are available, the incoming frames are discarded. If the frame was sent by TCP, TCP will resend the packet when a time-out is received. If UDP sent the data, it is up to the application to verify that the packets have been sent.

See Chapter 5, “Adapter tuning” on page 95 for a complete description of this mechanism and suggestions about tuning transmit and receive queues in the adapters. Also, a list with the most typical adapter used in SP systems and its specifications about adapter queues (default values and ranges).

9.3 ARP cache tuning

Address Resolution Protocol (ARP) translates dynamic Internet addresses for IP into the unique hardware MAC addresses for all adapters on local links to a node. Configurations larger than 128 nodes in your RS/6000 SP environment could have performance problems due to the size of the ARP cache when communicating using IP based protocols.

In the second part of this book you had a complete explanation of ARP protocol and ARP cache tuning (Section 4.2.7, “Address resolution” on page 40) and the description of the parameters needed to do this (Section 4.3.2, “AIX tunables” on page 46). At this point, we only give some recommendations about the best way to size ARP cache in SP systems.

- For systems with greater than 150 nodes, round the number of nodes down to the next power of 2, and use that for arptab_nb. Table 28 shows these values for systems from 1 to 512 nodes.

Table 28. Determining ARP tuning settings based on the number of nodes

Number of nodes	arptab_nb value
1-64	25(default)
65-128	64
129-256	128
257-512	256

- For nodes that have more than three network adapters, set arptab_bsiz to 2 times the number of active IP interfaces. Table 29 lists the sizes of the arptab_bsiz value, based on the number of IP interfaces.

Table 29. Determining ARP tuning settings based on number of IP interfaces

Number of interfaces	arptab_bsiz value
1-3	7
4	8
5	10
6	12
7	14
8 or more	2 x number of interfaces

9.4 NFS performance in SP system

The first problem related to NFS that we generally see in large SP system configurations occurs when a single node is acting as the NFS server for a large number of nodes. In this scenario, the aggregate number of NFS requests can overwhelm the NFS socket or nfsd daemons on the server, or enough daemons cannot be configured.

For the server configuration, the number of nfsd daemons are the primary concern. If you have a 64 node configuration and configured one NFS server for the other 63 nodes, and if all 63 client nodes made an NFS request at the same time, you will need at least 63 nfsd daemons on the server. If you had only eight nfsd daemons, as in the default configuration, then 55 NFS requests would have to wait and they could time out. Obviously, average NFS response time will be very slow.

However, there is a limit on how many nfsd daemons you can configure before the amount of processing on behalf of the NFS traffic overwhelms the server. Generally, when you configure more than 100 nfsd daemons on a uniprocessor, and 200 nfsd daemons on a SMP node, you start to see NFS performance degradation, depending on the characteristics of the NFS traffic. The size of the requests, the mix of NFS operations, and the number of processes on each node that generates NFS requests influence the amount of NFS performance degradation.

Determining the number of biod daemons to configure on the client node is a bit complicated. The limitation of six biod daemons on a client node per NFS mounted file system is imposed by NFS. If there is more than one NFS mounted file system on your client nodes, and the number of biod daemons gets too high, performance will suffer in the same way as when too many nfsd daemons are configured on the server. Again, a general rule is that more than 100 biod daemons on a uniprocessor and 200 biod daemons on a SMP node will start to degrade performance.

Note

When you configure more than 100 nfsd daemons on a uniprocessor system or more than 200 nfsd daemons on a SMP node, you start to see NFS performance degradation, depending on the characteristics of the NFS traffic. The same rule can be applied to biod daemons

If you already have a lot of nfsd daemons configured on the server, the best solution is to split the NFS server duties up across several nodes, and keep the potential number of concurrent NFS requests below 100.

See Section 4.5, “NFS tuning” on page 68 for a complete NFS mechanism description and more information about nfsd and biod daemons tuning.

9.5 External server considerations

Connections through external networks to other machines can restrict performance on certain nodes on the SP system. For example, some external servers may not support RFC1323 extension, and can threfore only use a maximum TCP/IP window of only 65536 to transfer data to or from a node. This causes different performance problems on connections to these servers from other connections on the same node to other SP nodes.

In some cases, external servers have to communicate with additional external servers. When such external servers establish a connection, the negotiated tunables may be set to small values. For example, the TCP/IP window size value is set to the least common denominator. In the case of an external server having a tcp_recvspace of 16384, that is the TCP/IP window size that is used. Such a small TCP/IP window size provides slow throughput if the sending node is on the far side of a switch from a gateway node.

If traffic is routed through other external network routers, you may not be able to use the optimal maximum transmission unit (MTU) or tcp_mssdflt size for that network. If this happens, adjust the tcp_sendspace and tcp_recvspace values accordingly.

To find the optimal tcp_sendspace and tcp_recvspace sizes, get the largest MTU size that the router will handle to other networks of the same type. To get the optimal tcp_sendspace and tcp_recvspace sizes for single-stream transfers, use the formula shown in Figure 23.

t = m * q, where: t = optimal tcp_sendspace and tcp_recvspace sizes m = largest MTU size that the router will handle to other networks q = smaller of the transmit queue and receive queue size for the adapter

Figure 23. Calculating tcp send/receive space sizes

This does not apply to the SP switch or SP switch2 because the switch adapter does not have transmit and receive queues. See Section 5.2, “SP Switch and SP Switch2 adapter tuning” on page 99 for more information on tuning rpoolsize and spoolsize.

The number produced by the formula in Figure 23 is the largest that you can use for a single socket before the adapter drops packets and TCP/IP has to do a retry on the dropped packets. If you have more than one active socket, then the size calculated using this formula needs to be divided by the number of active sockets. You need to avoid dropping packets on the adapter to get optimal throughput performance.

When configuring the nodes, you must plan for the amount of mbuf space for all interfaces to a node. An indication that the mbuf allocation is too small is getting a requests for mbufs denied count in the `netstat -m` output, but you need to set extendednetstats to 1. See Section 11.1, “IBM AIX Monitoring tools and commands” on page 243 for more information about the command.

```
# no -o extendednetstats
extendednetstats = 1

# netstat -m
2 mbuf cluster pages in use
7 Kbytes allocated to mbufs
0 requests for mbufs denied
0 calls to protocol drain routines
0 sockets not created because sockthresh was reached
```

Some of these problems cannot be overridden using tunables on the node, but need to be considered when tuning a node that communicates to outside servers. There are two network tunables that may help in solving the small packet problem to external servers. These `no` settings are:

- tcp_pmtu_discover
- udp_pmtu_discover

By setting these tunables to 1, when a connection is established to a connection on a remote network (if the external server or workstation supports MTU path discovery), the connection will determine the largest segment size that it can send without fragmentation. This eases the problem of setting tcp_mssdflt to a compromise value.

9.6 Single-server multiple-client node problems

Some application configurations, such as NFS, consist of a parent or server node with multiple client or remote nodes. Such applications have a potential for the client nodes, through a large TCP window size, to send large volumes of data to one server node using the nonblocking switch network, whereas the server node cannot handle the total traffic from the client nodes, due to demands on the server's mbuf pools. The dropped packets will be reflected in a large number of failures in `netstat -m` on the server node.

To further illustrate, if you had 64 client nodes, as shown in Figure 24 on page 198, with a TCP window size of 512 KB, the server node would need buffer space of 32 MB just to accommodate the incoming packets from the client nodes, all other connections and traffic aside. To determine the server node mbuf requirements, get the number of client nodes that will simultaneously send data to the server node, multiply by the TCP window size on each client node, then again multiply by the number of sockets that each client opens on the server. If the server has multiple roles, add additional mbuf space as required. You can see how easy it can be to exceed the maximum amount of memory that can be allocated for the network (thewall) or even the limitations of the server node itself, which may only contain 128 MB of memory. To prevent this scenario from happening, you must reduce the combined amount of data arriving from the client nodes to the server nodes.

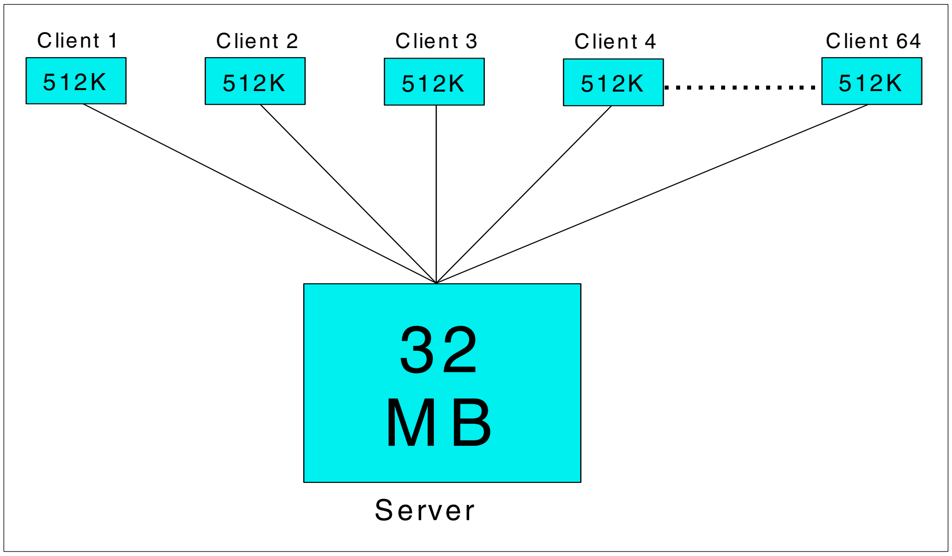

Figure 24. Single-server multiple-client scenario

You can use two methods to accomplish this:

- The first is to try to restrict the client nodes by setting tcp_sendspace and tcp_recvspace small enough so that the combined data sent by the client nodes does not exceed the buffer space on the server. While this reduces the receive buffer space required on the server, if that traffic must then be redirected to a terminal or remote file, you need to double the mbuf requirements to accommodate sending it back out again. If there are other applications on the client nodes that require the high switch bandwidth, they will not get the same throughput and may suffer due to the smaller setting for tcp_sendspace and tcp_recvspace. Traffic will back up and applications may slow down. It is a tradeoff of large data transfers on the client versus mbuf allocation on the server. To determine what values you should assign to tcp_sendspace and tcp_recvspace, first select the maximum number of buffers to allocate in the servers' mbuf pool, divide by the number of client nodes, and multiply by the average message size to get your window size. Set tcp_sendspace and tcp_recvspace on both the client and server side.
- The second method would be to restrict tcp_recvspace on the server node. The TCP window is negotiated upon establishment of the connection; the window value will be set to the lesser of tcp_sendspace and tcp_recvspace and will only impact client to server connections.

- If at all possible, the best solution would be to set the socket window sizes from within an application, leaving the tcp_sendspace and tcp_recvspace settings alone. This can be done by using the setsockopt() call within the application to change the values for SO_SNDBUFF and SO_RCVBUFF. Changing these values affects only the socket that the setsockopt() call was made against. All other connections would use their own setsockopt() call settings or the tcp_sendspace and tcp_recvspace settings.

Keep in mind that UDP has no mechanism for windowing and can have a greater amount of outstanding I/O on the connection. UDP should not be used to send large amounts of data over the switch because you could deplete the servers' mbuf pools very rapidly. Also be aware of very small messages; the minimum allocation from the mbuf pool is 256 bytes, so you could chew up all the mbuf space with a small amount of data.

An example of how a typical program would be modified to handle the setsockopt() call is shown in Figure 25. You can get more information in changing socket setting in the documentation for the setsockopt() call in the *AIX Technical Reference Volume 1*, SN32-9029.

```
/*We are the client if transmitting*/
if(options) {
   if(setsockopt(fd,SOL_SOCKET, options, &one, sizeof(one)) <0
                             err("setsockopt");
}
if(nodelay) {
   if(setsockopt(fd,IPPROTO_TCP,TCP_NODELAY,&one, sizeof(one)) <0
                             err("nodelay");
}
if(rfc1323) {
   if(setsockopt(fd,IPPROTO_TCP,TCP_RFC1323,&one, sizeof(one)) <0
                             err("rfc1323");
}
if (setsockopt(fd,SOL_SOCKET,SO_SNDBUF,&window,sizeof(window)) <0)
                        err("setsendwindow");
if (setsockopt(fd,SOL_SOCKET,SO_RCVBUF,&window,sizeof(window)) <0)
                         err("setreceivewindow");
if(connect(fd,&sinhim, sizeof(sinhim) ) < 0)
                err("connect");
mes("connect");
```

Figure 25. Sample setsockopt() call

9.7 Gateway or router node problems

Gateway or router nodes direct traffic between external networks and the SP system. Two different types of nodes are used as gateways: an existing SP node acting as a gateway, and the SP Switch Router, used as direct-attached SP Router node (also known as the GRF router node).

If a lot of traffic is routed through a router node, it affects any other job using that node. We suggest that router or gateway nodes not be assigned to high priority or fast response time parallel jobs unless there are no other nodes available, or the amount of expected network traffic is small.

When tuning a router or gateway, you need to plan on enough buffer space in the node to handle traffic on multiple interfaces. Traffic from the SP Switch uses the send and receive pools. Traffic to Ethernet, Token Ring and FDDI uses the system mbufs, while ATM uses its own buffer areas. Having a large difference between the switch pools and the amount of space for other adapter traffic leads to bottlenecks or dropped packets. The best initial settings are for the same amount of space for both sets of interfaces.

To optimize the packet size sent to remote networks, you may set the variable tcp_pmtu_discover = 1. This lowers the overhead on the gateway or router node. However, watch out for networks with more than a couple of hundred hosts. By turning tcp_pmtu _discover on, you are, in effect, creating a route in your routing table to every host that is out there. Any network greater than a couple of hundred hosts becomes very inefficient and performance problems will arise. Be sure to turn this variable on if your network has the correct number of hosts.

9.8 Typical performance problems by environments

Table 30 that summarizes the most common performance problems in SP systems, but categorized by environment.

Table 30. Common performance problems categorized by environment

Common to all applications	
What?	**Where?**
Nagle problem caused by incorrect TCP windows tuning	Go to Section 9.1, “The Nagle Algorithm” on page 191.
Single server / Multiple client scaling problems	Go to Section 9.6, “Single-server multiple-client node problems” on page 197.

Memory size problems	Go to Section 10.2, “Monitoring memory” on page 204
Commercial applications	
What?	**Where?**
SP switch pool sizing	Go to Section 5.2, “SP Switch and SP Switch2 adapter tuning” on page 99.
Excessive aggregate memory requirements	Go to Section 4.3, “AIX network tunables” on page 45.
Adapter queue overflows	Go to Section 9.2, “Adapter queue overflows” on page 192.
Scientific and technical applications	
What?	**Where?**
Synchronization point file server accesses	Go to Section 6.3, “I/O” on page 141.
File system performance	Go to Section 6.3, “I/O” on page 141.
Server environments	
What?	**Where?**
Exhaustion of switch pools due to use of large TCP windows	Go to Section 5.2, “SP Switch and SP Switch2 adapter tuning” on page 99.
Adapter queue overflows in non-switch adapters	Go to Section 5.3, “SP Ethernet tuning” on page 110.
NFS tuning (daemons & socket sizing)	Go to Section 4.5, “NFS tuning” on page 68.
Bottlenecks on the I/O bus or disks	Go to Section 6.3, “I/O” on page 141.
Buddy buffers sizing for VSD or GPFS	Go to Chapter 13, “VSD” on page 353 and Chapter 14, “GPFS” on page 369.

Chapter 10. Resource monitoring

In this chapter, we will discuss how to monitor system resources by using commands and tools that are available with the normal AIX distribution. Their use on the RS/6000 SP is the same as on any standalone RS/6000 multiprocessor (MP) or uniprocessor (UP) system. The main differences are the tools to monitor the SP Switch and SP Switch2.

10.1 IBM AIX monitoring tools and commands

AIX provides several monitoring tools to determine performance bottlenecks and to tune the system. The commands listed in Table 31 are useful for determining where bottlenecks are occurring on your system.

Table 31. IBM AIX monitoring tools by system resources

CPU	MEMORY	I/O	Network
topas	topas	topas	topas
vmstat	vmstat	vmstat	netstat
iostat	svmon	iostat	nfsstat
ps	filemon	fileplace	netpmon
sar	bf, bfrpt	filemon	iptrace, ipreport
netpmon	lsps	lspv, lslv, lsvg	tcpdump
pstat	crash		crash, ndb
acctcom	acctcom		nslookup, host, ntptrace, ntpq
			traceroute, ping
			col_dump
			entstat, tokstat, fddistat, atmstat, estat
trace, trcrpt	trace, trcrpt	trace, trcrpt	trace, trcrpt

10.2 Monitoring memory

To monitor the usage of memory in a SP node is very important, because a lack of memory will invariably lead to, at least, performance degradation, but in some cases even service discontinuance and application malfunctions.

The *default* values in /etc/security/limits, as shown here, will limit each process regarding the DATA, RSS and STACK properties of a user process, unless they are changed, which is often necessary.

```
# grep -p ^default /etc/security/limits
default:
        fsize = 2097151
        core = 2097151
        cpu = -1
        data = 262144
        rss = 65536
        stack = 65536
        nofiles = 2000
```

The limits that are highlighted above are *softlimits* that can be used to restrict how a user process is allowed to use system memory (real and virtual).

data — Identifies the soft limit for the largest process data segment for a user's process.

rss — Sets the soft limit for the largest amount of physical memory a user's process can allocate. This limit is not enforced by the system.

stack — Specifies the soft limit for the largest process stack segment for a user's process.

To enforce a *hardlimit*, the following attributes may be used instead:

data_hard — Identifies the largest process data segment for a user's process.

rss_hard — Sets the largest amount of physical memory a user's process can allocate. This limit is enforced by the system.

stack_hard — Specifies the largest process stack segment for a user's process.

The values are 512-byte blocks or -1 which indicates *unlimited*. If the hardlimits are not set, the data_hard, rss_hard and stack_hard will all default to unlimited.

10.2.1 System-wide monitoring

Memory monitoring can be done with many commands and tools. We will try to describe two starting points and how to pursue the resource specifics of memory monitoring.

10.2.1.1 To start with vmstat

The `vmstat` command shows how the system is performing its memory handling, as is shown in this example:

```
# vmstat 5
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b    avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0 2146520   130   0   4   3  67  238   0 237  243 222 31  7 30 32
 4 11 2146523   135   0   0   0 7008 25490   0 2868 8989 3711 24  9  0 67
 4  9 2145191   122   0   0   0 6812 50234   0 2746 4100 3362 27  8  0 65
 4  9 2145192   124   0   0   0 7602 48337   0 2766 3696 3606 20  8  0 72
 4 11 2145637   133   0   0   0 7109 46670   0 2777 8785 3479 25  8  0 67
 5 13 2145738   134   0   0   0 8267 66832   0 3600 22070 5037 40 12  0 48
 6 10 2144499   119   0   0   0 8348 83568   0 3435 31469 4695 33 14  0 53
 5 10 2144244   131   0   0   0 7325 53636   0 2943 26248 4871 30 12  0 58
 3 11 2144251   133   0   0   0 7309 44760   0 2987 16852 4465 16 12  0 71
<<< lines omitted >>>
```

To find out how much real memory this system has, you can use the `bosboot` and `lsattr` commands. Three examples of how to do it follows:

```
# bosboot -r
12582912

# lsattr -F value -El sys0 -a realmem
12582912

# lsdev -F name -Cc memory|grep ^mem|xargs -i lsattr -Fname:attribute:value -El {}
mem0:size:12288
mem0:goodsize:12288
```

The node has 12 GB of memory and it is running out of memory; `vmtune` can tell us what the VMM settings for this node are:

```
# /usr/samples/kernel/vmtune
vmtune:  current values:
  -p       -P        -r          -R         -f       -F       -N        -W
minperm  maxperm  minpgahead maxpgahead  minfree  maxfree  pd_npages maxrandwrt
 628939   943715       2           8        120      128     524288        0

  -M      -w      -k      -c        -b         -B           -u        -l
maxpin npswarn npskill numclust numfsbufs hd_pbuf_cnt lvm_bufcnt lrubucket defp
2516576  105984   26496       1      93        928          9      131072     1

        -s
sync_release_ilock
        0

number of valid memory pages = 3145719  maxperm=30.0% of real memory
maximum pinable=80.0% of real memory    minperm=20.0% of real memory
number of file memory pages = 1102288   numperm=35.0% of real memory
```

Just by looking at the previous `vmstat` output, we can almost certainly say that this 12 GB/12 CPU system is having a hard time, with a lot of threads blocked for I/O and a high wait for I/O. But the separation of system and user is OK even though there are a lot of system calls and context switches going on, and there is obviously a shortage of real memory. However, the `vmtune` output does show that someone has changed the default values for minperm/maxperm and that numperm is 35 percent. This would mean that approximately 4 GB is used for file caching. The system is running multiple ORACLE RDBMS instances and the disk layout is not optimized for speed.

Based on the information presented in the previous paragraph, we can make the following conclusions:

- The system is short on memory. This needs to be investigated further to determine if minperm/maxperm can be lowered substantially and maybe strict_maxperm can be used as well. Also, the minfree/maxfree should be increased, depending on the read behavior on the file systems. But for now, we will assume that we will make the following changes with `vmtune` to the system:
  - minfree to 240 and maxfree to 256
  - minperm to 3 and maxperm to 8
- There is a high percentage of I/O wait, but this is actually AIX V4.3.2, so it could be misleading (see Section 11.1.37, “vmstat” on page 324 for more information). But there are a lot of blocked threads, so we need to investigate the RDBMS layout as well. Usually an I/O intensive system can aggravate the memory usage, because more processes are simultaneously active during longer periods of time than would be the case if their I/O was dispensed with more quickly.

10.2.1.2 To start with svmon

The most memory monitoring oriented command, `svmon` will show how the system is performing it's memory handling a bit differently from `vmstat`, as is shown in this example:

```
# svmon -i 2 4|sed '/^$/d'
               size      inuse       free        pin    virtual
memory        65536      38389      27147       4599      49050
pg space     131072      26474
               work       pers       clnt
pin            4443        156          0
in use        25406      12983          0
               size      inuse       free        pin    virtual
```

The above example shows us that size of real memory frames are 65536 memory frames, 38389 of these frames contain pages, and 27147 are free. The size of the paging space is 131072 pages and 26474 are used.

.

Note

VMM manages virtual counter for statistics purpose. It may happen that all values are not updated at the same time.

10.2.1.3 To start with sar

Another good, but more versatile monitoring command, `sar` will also show how the system is performing its memory handling a bit differently from `vmstat`, as can be seen in this example:

```
# sar -qr 2 4|sed '/^$/d'
AIX night4n41 3 4 006001824C00    10/13/00
19:35:49 runq-sz %runocc swpq-sz %swpocc
           slots cycle/s fault/s  odio/s
19:35:54    70.0     100     2.0     100
         4191450    0.00   68.17   37.39
19:35:59    63.2     100     2.4     100
         4191504    0.00   33.80   41.79
19:36:04    55.8     100     2.8     100
         4192228    0.00  389.75   46.99
19:36:09    59.0     100     2.8     100
         4192229    0.00 9030.14   47.19
19:36:14    61.0     100     2.8     100
         4192229    0.00 9915.16   34.20
Average     61.8     100     2.6     100
Average  4191928       0    3887      42
```

The above example shows us that, on average, the node has 61.8 kernel threads in the run queue, that 2.6 kernel threads are waiting to be paged in. There are 3887 page faults per second (this is not a count of page faults that generate I/O, because some page faults can be resolved without I/O) and there are 42 non paging disk I/Os per second.

10.2.2 Application specific monitoring

To find out how much real memory an application uses, is important that the memory consumption of additional instances can be calculated. Each process has shared and non-shared parts. The non-shared are specific for each process, but the shared part can be used by other processes as well.

Since part of a process is usually shared, it is not so straightforward as to multiply one instance of a process memory consumption with the number of estimated instances of that process.

In a small scale system, this might not be a real issue, but counting hundreds and thousands of users this same way can give a large amount of perceived, but false, memory requirements. Say that an application uses 10 MB for the first instance; of these 10 MB, we know that 7 MB are shared. Then, by adding 500 users who use the same program, if we only calculated 10*500, we would have an overcapacity of 7*500 (3.5 GB) memory when it would be sufficient with approximately 3*500 1.5 GB.

It is important to find out how much memory is and will be used by adding more users and applications to a production system. To calculate the requirements for sizing memory for a particular application, we can use the following formula:

Total application size = shared size + (# processes * non-shared size)

This example uses the `svmon -dlC` command, which will show all processes running the specified application (note the highlighted process 16426, as this will be used in subsequent examples below).

```
# svmon -dlC ksh|sed '/^$/d'
===============================================================================
Command ksh               Inuse      Pin     Pgsp  Virtual
                           3495       19      172     4726
-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual   64-bit    Mthrd
   31094 ksh               2458        7       43     3509        N        N
<<< lines omitted >>>
   16426 ksh               2365        7       33     3442        N        N
...............................................................................
SYSTEM segments           Inuse      Pin     Pgsp  Virtual
                              6        6        0        7
<<< lines omitted >>>
...............................................................................
EXCLUSIVE segments        Inuse      Pin     Pgsp  Virtual
                           1172       13      146     1362
<<< lines omitted >>>
...............................................................................
SHARED segments           Inuse      Pin     Pgsp  Virtual
                           2317        0       26     3357
<<< lines omitted >>>
1e8de       - pers /dev/hd2:8280           1     0    -     -    0..0
                                          pid:39722    cmd: tn
                                          pid:35606    cmd: ksh
                                          pid:34248    cmd: ksh
                                          pid:32684    cmd: tn
                                          pid:31094    cmd: ksh
                                          pid:29056    cmd: smitty
                                          pid:28302    cmd: ksh
                                          pid:26964    cmd: ksh
                                          pid:26454    cmd: ksh
                                          pid:17628    cmd: ksh
<<< lines omitted >>>
```

Then `svmon` displays information about the segments used by those processes. This set of segments is separated into three categories:

- The segments that are flagged `SYSTEM`, that are basically shared by all processes.
- The segments that are only used by the set of processes (exclusive).
- The segments that are shared between several command names (shared).

Because we used the `-l` flag, then for each segment in the last category the list of process identifiers, that use the segment, are displayed. Beside the process identifier, the command name it runs is also displayed. Note that in our example above, the application ksh shares a segment with different applications as well (tn, smitty).

The first Inuse column indicates the total number of pages in real memory in segments that are used by all processes running the command.

By adding the Inuse columns from SYSTEM, EXCLUSIVE and SHARED we end up with the same amount as is shown in the Inuse column for the Command record (6+1172+2317 = 3495).

To examine a specific process, the following example shows how to use the -P option; this option also requires a process ID as parameter. In this example, we get the same data for this particular process as is highlighted in the previous example (2365 [2257+58+41+6+2+1]):

```
# svmon -P 16426|sed '/^$/d'
-------------------------------------------------------------------------------
     Pid Command        Inuse      Pin     Pgsp  Virtual   64-bit    Mthrd
   16426 ksh             2365        7       33     3442        N        N
  Vsid     Esid Type Description           Inuse   Pin Pgsp Virtual Addr Range
 13013        d work shared library text    2257     0   26  3357   0..65535
 18058        1 pers code,/dev/hd2:5382       58     0    -     -   0..58
 18ff8        2 work process private          41     1    6    76   0..82 :
                                                                    65310..65535
  e00e        0 work kernel shadow             6     6    0     7   0..12
 1c05c        - pers /dev/hd2:47253            2     0    -     -   0..1
 19cf9        f work shared library data       1     0    1     2   0..1850
  a0ab        - pers /dev/hd9var:203           0     0    -     -   0..0
  70a6        - pers /dev/hd3:80               0     0    -     -
```

Compare this with the output from the `ps` command:

```
# ps vax 16426
   PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
 16426      - A     0:00    0   500   400 32768   198   232  0.0  0.0 ksh
# ps lax 16426
     F S      UID   PID  PPID   C PRI NI ADDR  SZ  RSS   WCHAN    TTY  TIME CMD
240001 A        0 16426  9930   0  64 22 18ff8 696  400              -  0:00 ksh
```

A simpler way of looking at the output from `svmon` is as follows:

```
# svmon -C ksh|grep -p segments|sed '/^$/d;/^\./d'
SYSTEM segments         Inuse      Pin     Pgsp  Virtual
                            6        6        0        7
EXCLUSIVE segments      Inuse      Pin     Pgsp  Virtual
                         1176       13      144     1362
SHARED segments         Inuse      Pin     Pgsp  Virtual
                         2333        0       26     3357
```

10.2.2.1 To start with xmperf

Start xmperf on the CWS (or the system that PTX/6000 is installed on) and select **Monitor -> add new console**. Select the node IP address that is the subject of observation. In Figure 26 on page 211, we selected **Memory statistics -> physical memory** and then the values we want to look at. Here we have number of Computational pages resident in memory, Memory free, Memory pinned:

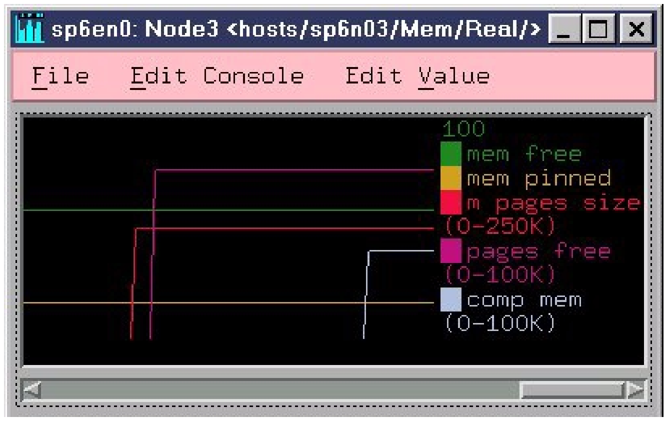

Figure 26. xmperf custom output for memory

10.2.3 User specific monitoring

To find out how much real memory a user is using for applications, the following is a way to find out.

This example uses the `svmon -dU` command, which will show all processes running for the specified user:

```
# svmon -dU root|sed '/^$/d'
===============================================================================
User root                  Inuse       Pin     Pgsp  Virtual
                           22413       294    10988    31048
-------------------------------------------------------------------------------
     Pid Command           Inuse       Pin     Pgsp  Virtual   64-bit    Mthrd
   29516 smitty             3365         7       26     4211        N        N
   29056 smitty             3342         7       96     4258        N        N
   18412 bin                3118         7      537     4415        N        N
   14192 bin                3072       165     1364     5208        N        Y
<<< lines omitted >>>
...............................................................................
SHARED segments            Inuse       Pin     Pgsp  Virtual
                            2263         0       26     3356
  Vsid      Esid Type Description              Inuse   Pin Pgsp Virtual Addr Range
 13013         d work shared library text       2261     0   26  3356   0..65535
  4204         1 pers code,/dev/hd2:509            2     0    -     -   0..40
<<< lines omitted >>>
```

10.3 Monitoring CPU

Monitoring the usage of CPU in a SP node is also very important, because if Workload Manager is not used, a process is usually allowed to utilize the available CPUs as much as its priority permits.

Applications that use spin-looping[1] and other such techniques, or that are just misbehaving, can cause system wide performance degradation (they are *CPU hogs*).

UNIX/AIX is different from other multi-user IBM systems, such as OS/400, because there is usually no limitations to how much resources a user application can make use of (except for what is defined in /etc/security/limits). The default values in /etc/security/limits, as shown here, will not limit the CPU time for processes:

```
# grep -p ^default /etc/security/limits
default:
        fsize = 2097151
        core = 2097151
        cpu = -1
        data = 262144
        rss = 65536
        stack = 65536
        nofiles = 2000
```

The limits that are highlighted above are *softlimits* that can be used to restrict how a user process is allowed to use system memory (real and virtual):

cpu — Sets the soft limit for the largest amount of system unit time (in seconds) that a user's process can use.

To enforce a *hardlimit*, the following attributes may be used instead:

cpu_hard — Sets the largest amount of system unit time (in seconds) that a user's process can use.

The values should be a decimal integer string representing the amount of system unit time in seconds or -1, which indicates *unlimited*. If the cpu_hard is not set, and cpu is unlimited, then cpu_hard by definition will be unlimited.

[1] Programming technique where a condition is tested and retested in a loop with, usually, a delay between tests.

10.3.1 System-wide monitoring

Memory monitoring can be done with many commands and tools; we will try to describe two starting points and how to pursue the resource specifics of memory monitoring.

10.3.1.1 To start with vmstat

The `vmstat` command shows how the CPU resources are used, as shown in the following example:

```
# vmstat 5
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b     avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0 2146520   130   0   4   3  67  238   0 237  243 222 31  7 30 32
 4 11 2146523   135   0   0   0 7008 25490   0 2868 8989 3711 24  9  0 67
 4  9 2145191   122   0   0   0 6812 50234   0 2746 4100 3362 27  8  0 65
 4  9 2145192   124   0   0   0 7602 48337   0 2766 3696 3606 20  8  0 72
 4 11 2145637   133   0   0   0 7109 46670   0 2777 8785 3479 25  8  0 67
 5 13 2145738   134   0   0   0 8267 66832   0 3600 22070 5037 40 12  0 48
 6 10 2144499   119   0   0   0 8348 83568   0 3435 31469 4695 33 14  0 53
 5 10 2144244   131   0   0   0 7325 53636   0 2943 26248 4871 30 12  0 58
 3 11 2144251   133   0   0   0 7309 44760   0 2987 16852 4465 16 12  0 71
<<< lines omitted >>>
```

As can be seen in the preceding example, the system has a good division between system (sy) and user (us) mode. But the waiting (wa) for I/O is very high, and the system has problems executing all I/O, as there are around ten kernel threads that are blocked waiting for resources or I/O (b). The run queue (r) is OK with around four kernel threads waiting to be dispatched.

But since we know that this is a 12 CPU node, the run queue (r) might even be a bit low to keep all CPUs occupied, especially with the high I/O wait (wa) and blocked waiting for resources or I/O (b). Therefore, we can use the `pstat` command to look at what is running on the different CPUs:

```
# pstat -S
STATUS OF PROCESSORS:

CPU     TID  TSLOT     PID  PSLOT  PROC_NAME
  0  25e5db   9701   18544    389       no  pstat
  1     307      3     306      3       no  wait
  2     e1d     14     e1c     14       no  lrud
  3  422879  16936   4268c   1062       no  oracle
  4  6e899d  28297   1e496    484       no  oracle
  5     70f      7     70e      7       no  wait
  6  55f4c3  22004   3a16a    929       no  oracle
  7     913      9     912      9       no  wait
  8     a15     10     a14     10       no  wait
  9     b17     11     b16     11       no  wait
 10  25e5db   9701   18544    389       no  pstat
 11     d1b     13     d1a     13       no  wait
```

As we suspected, the system has idle processors (where the wait processes are running); at this snapshot in time, only half (six) of the CPUs were busy.

Compare this with the following `pstat` snapshot from a benchmarking node running a CPU constrained application:

```
# pstat -S
STATUS OF PROCESSORS:

CPU     TID  TSLOT     PID  PSLOT  PROC_NAME
  0    f6fd    246   1a228    418       no  crash
  1    7771    119    501a     80       no  hxebench
  2    9c39    156    71e2    113       no  hxeflp
  3    c3d1    195    8000    128       no  hxehd
  4    6875    104    3a3e     58       no  hxebench
  5    618d     97    4728     71       no  hxebench
  6    3fef     63    4ee2     78       no  hxebench
  7    7efd    126    4ab2     74       no  hxebench
  8    7fff    127    51a4     81       no  hxebench
  9    a347    163    78f0    120       no  hxeflp
 10    8103    129    56ac     86       no  hxebench
 11    ab57    171    8000    128       no  hxehd
 12    8811    136    5dba     93       no  hxecolony
 13    a74f    167    7cf8    124       no  hxeflp
 14    a851    168    7dfa    125       no  hxeflp
 15    860d    134    5bb6     91       no  hxebench
```

Note that this is a 16 CPU system. The corresponding `vmstat` output for this system follows:

```
# vmstat 5 5
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 3  0 613796 861153   0   0   0   0    0   0 145   860 285 97  2  1  0
67  2 272830 1202381   0   0   0   0    0   0 2397 14261 4701 97  3  0  0
62  2 300855 1174452   0   0   0   0    0   0 2382 12615 4320 97  3  0  0
63  2 361943 1113393   0   0   0   0    0   0 2357 12922 4633 98  2  0  0
65  2 421961 1053406   0   0   0   0    0   0 2360 12963 4614 98  2  0  0
<<< lines omitted >>>
```

This node has no waiting for I/O (wa) and many kernel threads ready to run (r). However, the fact that the system is not idle waiting for I/O does not mean that there is no I/O; it just does not show up here.

10.3.1.2 To start with sar

The `sar` command will show how the CPU resources are used, as is shown in the following example. First, by looking at the mean values for all CPUs put together:

```
# sar 5 5

AIX night4n41 3 4 006001824C00    10/13/00

13:37:57    %usr    %sys    %wio   %idle
13:38:02      98       2       0       0
13:38:07      98       2       0       0
13:38:12      98       2       0       0
13:38:17      98       2       0       0
13:38:22      98       2       0       0

Average       98       2       0       0
```

Note that the different samples shown here were made during the same run, but not during the same time, as can be seen from the timestamps.

As can be seen above, we have extremely high utilization of the CPU resources in this system. The next example shows the same node, but with `sar` showing the statistics for each CPU:

```
# sar -P ALL 1 5

AIX night4n41 3 4 006001824C00    10/13/00

13:46:24 cpu    %usr    %sys    %wio   %idle
<<< lines omitted >>>
13:46:27  0      100       0       0       0
          1      100       0       0       0
          2      100       0       0       0
          3      100       0       0       0
          4       99       1       0       0
          5      100       0       0       0
          6       99       1       0       0
          7       98       2       0       0
          8      100       0       0       0
          9      100       0       0       0
          10       73      27       0       0
          11       98       2       0       0
          12      100       0       0       0
          13       98       2       0       0
          14       99       1       0       0
          15       99       1       0       0
          -       98       2       0       0
<<< lines omitted >>>
Average   0      100       0       0       0
```

Apart from the actual CPU usage, it is also important to understand how the scheduler queues are building up, as can be seen in this example:

```
# sar -q 2 4|sed '/^$/d'
AIX night4n41 3 4 006001824C00    10/13/00
19:35:49 runq-sz %runocc swpq-sz %swpocc
19:35:54    70.0     100     2.0     100
19:35:59    63.2     100     2.4     100
19:36:04    55.8     100     2.8     100
19:36:09    59.0     100     2.8     100
19:36:14    61.0     100     2.8     100
Average     61.8     100     2.6     100
```

The above example shows us that on average the node has 61.8 kernel threads in the run queue and that 2.6 kernel threads are waiting to be paged in every second.

The following shows the use of `vmstat` and `iosat` for monitoring the same node:

```
# vmstat 1 5;iostat -t 1 5
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 3  0 446108 1029240   0   0   0   0    0   0 145   858 284 97  2  1  0
63  4 456483 1018857   0   0   0   0    0   0 2255 18464 4260 97  3  0  0
71  2 466646 1008695   0   0   0   0    0   0 2188 14415 4477 97  3  0  0
59  2 477736 997604   0   0   0   0    0   0 2222 14623 4494 98  2  0  0
82  2 488600 986746   0   0   0   0    0   0 2181 15455 4578 98  2  0  0

tty:      tin         tout    avg-cpu:  % user    % sys     % idle    % iowait
          0.0          1.1               97.2      2.1        0.8       0.0
          0.0        145.7               97.9      2.1        0.0       0.0
          0.0         80.7               97.6      2.4        0.0       0.0
          0.0         80.8               97.9      2.1        0.0       0.0
          0.0         80.2               98.0      2.0        0.0       0.0
```

Finally, we look at the node with the `topas` command (`topas -n1 -d3 -p6`):

```
Topas Monitor for host:    night4n41          EVENTS/QUEUES    FILE/TTY
Fri Oct 13 13:57:05 2000   Interval:  2       Cswitch    4655  Readch 46228608
                                              Syscall   14726  Writech  141048
Kernel    2.1   |#                          | Reads      1499  Rawin        0
User     97.8   |########################### | Writes      259  Ttyout     163
Wait      0.0   |                           | Forks         0  Igets      475
Idle      0.0   |                           | Execs      66.5  Namei      530
                                              Runqueue   73.1  Dirblk       5
Interf   KBPS6     2.9  O-Pac4   KB-I2  KB-Ou4  Waitqueue   2.5
en0        0.7     3.4     3.4     0.2     0.5
                                              PAGING           MEMORY
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  Faults    10534  Real,MB    6143
hdisk0    0.0    328.2    73.4     0.0   328.2  Steals        0  % Comp     73.0
hdisk1    0.0    319.8    10.4   207.9   111.9  PgspIn        0  % Noncomp   1.0
                                              PgspOut       0  % Client    0.0
hxehd    (32768)123.2% PgSp: 3.0mb root        PageIn        0
hxebench (22446) 64.0% PgSp:20.7mb root        PageOut      82  PAGING SPACE
hxebench (20900) 59.2% PgSp:20.7mb root        Sios         73  Size,MB   16384
hxebench (22188) 57.2% PgSp:20.7mb root                        % Used      0.0
hxebench (8514)  57.2% PgSp:20.7mb root                        % Free     99.9
hxebench (20454) 57.2% PgSp:20.7mb root

                                                 Press "h" for help screen.
                                                 Press "q" to quit program.
```

10.3.1.3 To start with xmperf and 3dmon

With xmperf and 3dmon, it is possible to view a 3D monitor. Just select Utilities in the xmperf window and open a 3dmon window. You can select the preferred output. In our example, shown in Figure 27 on page 218, we observe the CPU 0-3.

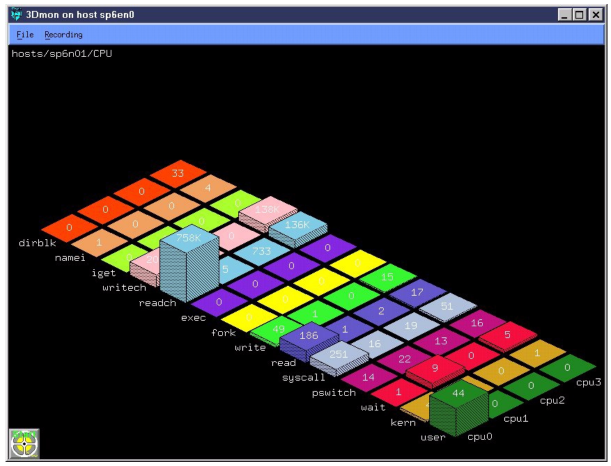

Figure 27. 3dmon CPU monitor

10.3.2 Application specific monitoring

To examine the CPU usage of a specific running process, the `ps` command is usually used. But as we will also show, `crash` and `pstat` can also be used.

But first, we will run `topas -n0 -d0` to get a rough feeling for the overall performance and the top CPU using processes:

```
Topas Monitor for host:    night4n41           EVENTS/QUEUES    FILE/TTY
Fri Oct 13 20:07:29 2000   Interval:  2        Cswitch    4523  Readch  6593122
                                               Syscall   15066  Writech 3261987
Kernel     2.1   |#                          | Reads      1688  Rawin         0
User      97.8   |###########################| Writes      316  Ttyout      623
Wait       0.0   |                           | Forks         0  Igets         0
Idle       0.0   |                           | Execs         0  Namei       615
                                               Runqueue   62.5  Dirblk        4
hxebench (23478) 55.1% PgSp:20.7mb root        Waitqueue   2.1
hxebench (22446) 55.1% PgSp:20.7mb root
hxebench (14910) 54.9% PgSp:20.7mb root        PAGING           MEMORY
hxebench (20506) 54.4% PgSp:20.7mb root        Faults    11597  Real,MB    6143
hxebench (18216) 54.4% PgSp:20.7mb root        Steals        0  % Comp     63.0
hxebench (20454) 54.1% PgSp:20.7mb root        PgspIn        0  % Noncomp   1.0
hxebench (19122) 53.8% PgSp:20.7mb root        PgspOut       0  % Client    0.0
hxebench (20194) 52.7% PgSp:20.7mb root        PageIn        0
hxebench (21930) 52.1% PgSp:20.7mb root        PageOut      39  PAGING SPACE
hxeflp   (30960) 46.9% PgSp: 0.4mb root        Sios         35  Size,MB   16384
                                                                % Used      0.0
                                                                % Free     99.9
```

From the `topas` screen, we decide to find out more about these hxebench processes that are using most of the CPU resources. First, we use `ps` with the lax option (the w option is only used to get longer lines from `ps`):

```
# ps laxw|egrep 'PID|hxebench'
      F S      UID    PID   PPID   C PRI NI ADDR  SZ  RSS   WCHAN    TTY  TIME CMD
 240001 A        0   8514  17148  66 126 38 88551 21484 21612            - 1556:20 hxe
 240001 A        0  14910  17148  66 126 38 a8555 21484 21612            - 1553:28 hxe
 240001 A        0  15556  17148   6 100 38 40548 21484 21612            - 177:01 hxeb
 240001 A        0  18216  17148  64 126 38 c8559 21484 21612            - 1539:32 hxe
 240001 A        0  19122  17148  68 126 38 8561 21484 21612            - 1508:06 hxeb
 240001 A        0  20194  17148  41 126 38 e855d 21484 21612            - 1514:37 hxe
 240001 A        0  20454  17148  65 126 38 6854d 21484 21612            - 1570:16 hxe
 240001 A        0  20506  17148  41 126 38 10542 21484 21612            - 1587:40 hxe
 240001 A        0  20900  17148  42 126 38 28565 21484 21612            - 1479:48 hxe
 240001 A        0  21930  17148  68 126 38 48569 21484 21612            - 1450:17 hxe
```

From the output above, we get priority, nice value and how much CPU time the process has accumulated. However, we also get a lot of unnecessary information, so we make our own selection using the -F option to `ps`:

```
# ps -ea -F "pid stat cpu pri nice sched bnd pcpu time comm"|egrep "PID|hxebench"
   PID ST  CP PRI NI SCH BND  %CPU       TIME COMMAND
  8514 A   62 126 38   0   3   3.0  1-02:12:18 hxebench
 14910 A   61 126 38   0   4   3.0  1-02:10:52 hxebench
 15556 A    6 100 38   0   0   0.3    02:58:59 hxebench
 18216 A   62 126 38   0   5   3.0  1-01:56:30 hxebench
 19122 A   39 126 38   0   7   2.9  1-01:25:53 hxebench
 20194 A   62 126 38   0   6   2.9  1-01:31:10 hxebench
 20454 A   61 126 38   0   2   3.1  1-02:27:45 hxebench
 20506 A   40 126 38   0   1   3.1  1-02:45:31 hxebench
 20900 A   64 126 38   0   8   2.9  1-00:56:46 hxebench
 21930 A   38 125 38   0   9   2.8  1-00:25:58 hxebench
 22188 A   38 125 38   0  10   2.7    23:43:58 hxebench
 22446 A   38 125 38   0  11   2.7    23:24:36 hxebench
 22704 A   60 126 38   0  12   2.6    22:47:21 hxebench
 22962 A   58 126 38   0  13   2.5    21:55:49 hxebench
 23220 A   38 125 38   0  14   2.4    20:57:01 hxebench
 23478 A   62 126 38   0  15   2.4    20:29:39 hxebench
```

Now we get process status (ST), CPU utilization (CP), priority (PRI), nice value (NI), scheduling policy (SCH), processor binding (BND), percent CPU time (%CPU), and accumulated CPU time (TIME).

An explanation for the output of our selection follows:

ST Contains the state of the process:

0 Nonexistent

A Active

I Intermediate

Z Canceled

T Stopped

K Available kernel process

CP CPU utilization of process or thread, incremented each time the system clock ticks and the process or thread is found to be running. The value is decayed by the scheduler by dividing it by 2, once per second. For the sched_other policy, CPU utilization is used in determining process scheduling priority. Large values indicate a CPU intensive process and result in lower process priority, whereas small values indicate an I/O intensive process and result in a more favorable priority.

PRI The priority of the process or kernel thread; higher numbers mean lower priority.

NI The nice value; used in calculating priority for the sched_other policy.

SCH The scheduling policy for a kernel thread. The policies are displayed using integers starting from 0. See Chapter 6, “Tuning other components in AIX” on page 115.

BND The logical processor number of the processor to which the kernel thread is bound (if any). For a process, this field is shown if all its threads are bound to the same processor.

%CPU The percentage of time the process has used the CPU since the process started. The value is computed by dividing the time the process uses the CPU by the elapsed time of the process. In a multi-processor environment, the value is further divided by the number of available CPUs, because several threads in the same process can run on different CPUs at the same time. (Because the time base over which this data is computed varies, the sum of all %CPU fields can exceed 100%.)

TIME The total execution time for the process.

We now use the same options to create a list of the ten highest CPU-intensive processes on the node:

```
# print " PID ST  CP PRI NI SCH BND  %CPU       TIME COMMAND"
# ps -ea -F "pid stat cpu pri nice sched bnd pcpu time comm"|tail +2|sort -k8nr|head -1
   PID ST  CP PRI NI SCH BND  %CPU       TIME COMMAND
 20454 A  101 126 38   0   2   3.1  1-02:31:11 hxebench
 20506 A   62 126 38   0   1   3.1  1-02:48:23 hxebench
  8514 A  100 126 38   0   3   3.0  1-02:15:43 hxebench
 14910 A   68 126 38   0   4   3.0  1-02:14:23 hxebench
 18216 A   97 126 38   0   5   3.0  1-01:59:40 hxebench
 19122 A   94 126 38   0   7   2.9  1-01:29:00 hxebench
 20194 A   95 126 38   0   6   2.9  1-01:34:16 hxebench
 20900 A   99 126 38   0   8   2.9  1-00:59:38 hxebench
 21930 A  102 126 38   0   9   2.8  1-00:29:24 hxebench
 22188 A  100 126 38   0  10   2.7    23:46:55 hxebench
```

The following example shows that the application spinner occupies the top five positions when it comes to CPU utilization in this node. It also shows us that the application is not paging, but it has a short execution time (TIME), and because since it is on the top of the list, it has a huge CPU usage (%CPU).

```
# ps gvc|head -1;ps gvc|tail +2|sort -rk11|head -5
   PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
  34910  pts/6 A     0:20    0   296   424 32768   197   228 48.8  0.0 spinner
  42064  pts/6 A     0:08    0   300   424 32768   197   228 29.6  0.0 spinner
  37196  pts/6 A     0:01    0   316   424 32768   197   228 25.0  0.0 spinner
  39602  pts/6 A     0:03    0   304   424 32768   197   228 20.0  0.0 spinner
  40112  pts/6 A     0:02    0   308   424 32768   197   228 18.2  0.0 spinner
```

10.4 Monitoring disk I/O

To monitor the disk I/O usage in a SP node is also very important, because this is the slowest storage resource for online operations. If not customized properly, disk I/O will lead to performance degradation, and in some cases severe performance degradation.

10.4.1 System-wide monitoring

A node that shows I/O waits has a problem when the CPU has very little idle time, in which case CPU utilization is constrained by the I/O subsystem.

However, there are often I/O waits with a lightly loaded system. As there are only a few processes executing, the scheduler is unable to keep the CPU(s) utilized. Environments with few active processes and significant I/O waits require application changes. Changing an application to use asynchronous I/O is often effective.

As the number of processes executing on a system increases, it is easier for the scheduler to find dispatchable work. In this case, disk I/O wait will become less noticeable, even though the system might perform disk I/O.

10.4.1.1 To start with iostat

To monitor disk I/O, we will usually start with the `iostat` command. The `iostat` command will show in great detail the load on different logical disks. The output below is the summary since boot time (if the `iostat` attribute has been enabled for the `sys0` logical device driver):

```
# iostat  -d 5
                " Disk history since boot not available. "


Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0          17.4     122.8     30.1          0       620
hdisk1          81.0     465.1     63.4          0      2349
hdisk0          20.0     145.9     36.3          0       731
hdisk1          82.8     461.1     63.1          0      2310
hdisk0          19.0     132.0     31.0          0       660
hdisk1          80.4     493.3     64.8          0      2467
hdisk0          20.4     148.3     34.6          0       742
hdisk1          84.6     476.6     64.6          0      2384
hdisk0          24.6     164.8     38.0          0       824
hdisk1          79.4     467.3     63.6          0      2337
hdisk0          24.0     173.6     40.6          0       868
hdisk1          80.2     490.3     64.6          0      2452
```

As can be seen in the output above, the `iostat` attribute has not been enabled for sys0, hence the “Disk history since boot not available message". This node only has two disks and one is 20 percent used while the other is 80 percent used. We need to find out more about those disks; what is on them and what they are used for. Here are a few of the actions we need to perform:

- We need to check what volume group the disks in question belongs to.
- We need to check the logical volume layout on the disks in question.
- We need to check the logical volume layout on all the disks in question in the volume group.

To accomplish these tasks, we will use the `lsvg`, `lspv` and `lslv` commands. First, let us find out what volume groups the disks belong to:

```
# lspv
hdisk0          00601452e0737ca0    rootvg
hdisk1          00601452e496116e    rootvg
```

They both belong to the rootvg volume group (as could be expected in this case). Since the two disks in question belongs to the same volume group, we can go ahead and list some information about the disks from the volume group perspective with `lsvg`:

```
# lsvg -p rootvg
rootvg:
PV_NAME           PV STATE     TOTAL PPs    FREE PPs    FREE DISTRIBUTION
hdisk0            active       271          0           00..00..00..00..00
hdisk1            active       542          2           00..00..00..00..02
```

We now see that the disks have different number of physical partitions, and since volume groups have one partition size, they must be of different size. Let us check the volume group to find out the PP size:

```
# lsvg rootvg
VOLUME GROUP:   rootvg                   VG IDENTIFIER:  00600245fb9a9061
VG STATE:       active                   PP SIZE:        32 megabyte(s)
VG PERMISSION:  read/write               TOTAL PPs:      813 (26016 megabytes)
MAX LVs:        256                      FREE PPs:       2 (64 megabytes)
LVs:            12                       USED PPs:       811 (25952 megabytes)
OPEN LVs:       10                       QUORUM:         2
TOTAL PVs:      2                        VG DESCRIPTORS: 3
STALE PVs:      0                        STALE PPs:      0
ACTIVE PVs:     2                        AUTO ON:        yes
MAX PPs per PV: 1016                     MAX PVs:        32
```

The PP size is 32 MB, so hdisk0 and hdisk1 are 8 GB and 17 GB disks respectively (approximate). Let us find out what logical volumes occupy the rootvg volume group:

```
# lsvg -l rootvg
rootvg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
hd5                 boot       1     1     1    closed/syncd  N/A
hd6                 paging     288   288   2    open/syncd    N/A
hd8                 jfslog     1     1     1    open/syncd    N/A
hd4                 jfs        8     8     1    open/syncd    /
hd2                 jfs        77    77    1    open/syncd    /usr
hd9var              jfs        2     2     1    open/syncd    /var
hd3                 jfs        1     1     1    open/syncd    /tmp
hd1                 jfs        1     1     1    open/syncd    /home
lv00                sysdump    10    10    1    closed/syncd  N/A
hd7                 sysdump    89    89    1    open/syncd    N/A
paging00            paging     224   224   1    open/syncd    N/A
lv0hd0              jfs        109   109   1    open/syncd    N/A
```

We cam see that there is no mirroring and that lv00 is not used and could be removed (this does not matter performance-wise), but there are two paging spaces (hd6 and paging00), and lv0hd0 has no file system mount point in /etc/filesystems (we will get back to this logical volume further down).

We now need to get more information on the layout on the disks. The following is the output from `lspv` to find out the intra-disk layout of logical volumes on hdisk1 (the 17 GB disk with 80 percent utilization):

```
# lspv -l hdisk1
hdisk1:
LV NAME              LPs   PPs   DISTRIBUTION          MOUNT POINT
lv0hd0               109   109   10..00..00..00..99    N/A
hd6                  108   108   99..09..00..00..00    N/A
lv00                 10    10    00..10..00..00..00    N/A
hd7                  89    89    00..89..00..00..00    N/A
paging00             224   224   00..00..108..108..08  N/A
```

As shown above, there are no file systems on this disk, and the only application logical volume is lv0hd0. The layout of logical volumes on the disk is not especially good, but we have also two paging spaces (hd6 and paging00) on the same disk, which is not good either. The lv0hd0 logical volume is also not allocated over one area, but is divided in two parts, one at the edge of the disk and one at the center. This is, of course, not good either. Let us view the intra-disk layout in another, more readable, way with the `lspv` command:

```
# lspv -p hdisk1
hdisk1:
PP RANGE  STATE   REGION        LV NAME            TYPE       MOUNT POINT
  1-10    used    outer edge    lv0hd0             jfs        N/A
 11-109   used    outer edge    hd6                paging     N/A
110-119   used    outer middle  lv00               sysdump    N/A
120-208   used    outer middle  hd7                sysdump    N/A
209-217   used    outer middle  hd6                paging     N/A
218-325   used    center        paging00           paging     N/A
326-433   used    inner middle  paging00           paging     N/A
434-441   used    inner edge    paging00           paging     N/A
442-540   used    inner edge    lv0hd0             jfs        N/A
541-542   free    inner edge
```

The highlighted partitions, in both the output examples above, for logical volume lv0hd0 shows the same information; it has its large part on the *inner edge* and 10 percent on the *outer edge*. This means that the disk arms have to move across the disk platter whenever the end of the first part of the logical volume. Let us examine how the logical volumes partitions are organized with the `lslv` command:

```
lv0hd0:N/A
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0001 hdisk1
0002  0002 hdisk1
0003  0003 hdisk1
0004  0004 hdisk1
0005  0005 hdisk1
0006  0006 hdisk1
0007  0007 hdisk1
0008  0008 hdisk1
0009  0009 hdisk1
0010  0010 hdisk1
0011  0442 hdisk1
0012  0443 hdisk1
<<< lines omitted >>>
0108  0539 hdisk1
0109  0540 hdisk1
```

The output just confirms that the first ten partitions (and 10 percent of its size) are on the *outer edge* and the rest on the *inner edge* for the `lv0hd0` logical volume. If we want LVMs opinion on whether it is a good layout or not, we cam run the following `lslv` command and check the IN BAND percentage:

```
# lslv -l lv0hd0
lv0hd0:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk1            109:000:000   0%            010:000:000:000:099
```

LVM agrees with us that this is not a good intra-disk layout for this logical volume. Let us check the paging spaces with the `lsps` command before we continue:

```
# lsps -a
Page Space  Physical Volume   Volume Group    Size   %Used  Active  Auto  Type
paging00    hdisk1            rootvg       7168MB      1     yes   yes    lv
hd6         hdisk0            rootvg       5760MB      1     yes   yes    lv
hd6         hdisk1            rootvg       3456MB      1     yes   yes    lv

# lsps -s
Total Paging Space   Percent Used
      16384MB              1%
```

This shows us that there is 16 GB of paging space allocated, but only 1 percent used (approximately 165 MB). So, we now need to know how much memory we have in the node:

```
# bootinfo -r
6291456
```

Since the system is using the Deferred Page Space Allocation (see Section 6.1, “Memory” on page 115), we will delete paging00 and shrink hd6 to maybe 1 to 2 GB.

Let us continue with examining the layout. Let us now check the other disk:

```
# lspv -p hdisk0
hdisk0:
PP RANGE  STATE   REGION        LV NAME             TYPE       MOUNT POINT
  1-1     used    outer edge    hd5                 boot       N/A
  2-55    used    outer edge    hd6                 paging     N/A
 56-109   used    outer middle  hd6                 paging     N/A
110-110   used    center        hd8                 jfslog     N/A
111-111   used    center        hd4                 jfs        /
112-121   used    center        hd2                 jfs        /usr
122-122   used    center        hd9var              jfs        /var
123-123   used    center        hd3                 jfs        /tmp
124-124   used    center        hd1                 jfs        /home
125-157   used    center        hd2                 jfs        /usr
158-163   used    center        hd6                 paging     N/A
164-209   used    inner middle  hd6                 paging     N/A
210-212   used    inner middle  hd2                 jfs        /usr
213-213   used    inner middle  hd9var              jfs        /var
214-217   used    inner middle  hd4                 jfs        /
218-220   used    inner edge    hd4                 jfs        /
221-251   used    inner edge    hd2                 jfs        /usr
252-271   used    inner edge    hd6                 paging     N/A
```

The layout was not good on this disk either. The hd6 paging space occupies this disk as well, and the logical volumes are spread out over the disk area. This is usually the case when file systems are expanded during production.

This is an excellent feature of AIX LVM, but after a while, logical volumes need to be reorganized so that they occupy consecutive physical partitions.

Returning to the logical volume lv0hd0, it seems to be the only application logical volume on the node. Is it used as a raw logical volume and, if so, in what way? Let us first examine the logical volume with `lslv`:

```
# lslv lv0hd0
LOGICAL VOLUME:     lv0hd0                 VOLUME GROUP:   rootvg
LV IDENTIFIER:      00600245fb9a9061.12    PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       opened/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            125                    PP SIZE:        32 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                109                    PPs:            109
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: on
EACH LP COPY ON A SEPARATE PV ?: yes
```

It looks OK, but if we try `getlvcb` on the logical volume, we get the following output:

```
# getlvcb -AT lv0hd0

         intrapolicy = &
         copies = 40
         interpolicy =
         lvid = MDHF…d
         lvname =
         label =
         machine id = #
         number lps = 41
         relocatable = '
         strict = (
         stripe width = 106
         stripe size in exponent = 105
         type = v0hd0
         upperbound = 39
         fs =
         time created  =         time modified =
```

This does not appear to be a normal logical volume, so let us try and read the first part of the logical volume with `dd` to confirm that this logical volume writes from byte 0 and does not leave any space in the beginning of the logical volume for the logical volume control block (LVCB):

```
# dd if=/dev/lv0hd0 bs=16 count=1 2>/dev/null|od -C
0000000   \0  \0  \0  \0   9   è   Ê 021   r   l   v   0   h   d   0  \0
0000020
```

This logical volume is completely overwritten; compare the preceding output to the output from /usr (hd2):

```
# dd if=/dev/hd2 bs=16 count=1 2>/dev/null|od -C
0000000    A   I   X       L   V   C   B  \0  \0   j   f   s  \0  \0  \0
0000020
```

10.4.1.2 To start with filemon

Monitoring disk I/O with the `filemon` command is usually done when there is a known performance issue with regards to the I/O. The `filemon` command will show, in great detail, the load on different disks, logical volumes and files:

```
# filemon -o filemon.out -O all && sleep 180;trcstop
```

The output from the `filemon` command can be quite extensive. To quickly find out if something is in need of attention, run the following `awk` commands to extract specific summary tables from the complete filemon output file. In the first report extraction, we want to look at disk I/O:

```
# awk '/Most Active Physical Volumes/,/^$/' filemon.out
Most Active Physical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume               description
------------------------------------------------------------------------
  0.24     16  50383  171.9  /dev/hdisk0          N/A
  0.08  68608  36160  357.4  /dev/hdisk1          N/A
```

This shows us that hdisk1 is more active than hdisk0 with almost twice the amount of transferred data (KB/s). We could now examine the disks, volume groups and logical volumes, as done in the previous section.

To get more detailed information on the usage of the logical volumes, extract the “Most Active Logical Volumes” part from the output file:

```
# awk '/Most Active Logical Volumes/,/^$/' filemon.out
Most Active Logical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume               description
------------------------------------------------------------------------
  0.22      0  37256  127.1  /dev/hd8             jfslog
  0.08  68608  36160  357.4  /dev/lv0hd0          N/A
  0.04      0  11968   40.8  /dev/hd3             /tmp
  0.01      0    312    1.1  /dev/hd4             /
  0.01     16    536    1.9  /dev/hd2             /usr
  0.00      0    311    1.1  /dev/hd9var          /var Frag_Sz.= 512
```

The logical volume lv0hd0 is by far the most utilized for both reading and writing, so now we extract information about this particular logical volume from the output file (since it appears in the summary part to have a detailed section as well):

```
# awk '/VOLUME: \/dev\/lv0hd0/,/^$/' filemon.out
VOLUME: /dev/lv0hd0  description: N/A
reads:                  1072    (0 errs)
  read sizes (blks):    avg    64.0 min      64 max      64 sdev     0.0
  read times (msec):    avg   7.112 min   2.763 max  29.334 sdev   2.476
  read sequences:       1072
  read seq. lengths:    avg    64.0 min      64 max      64 sdev     0.0
writes:                 565     (0 errs)
  write sizes (blks):   avg    64.0 min      64 max      64 sdev     0.0
  write times (msec):   avg   7.378 min   2.755 max  13.760 sdev   2.339
  write sequences:      565
  write seq. lengths:   avg    64.0 min      64 max      64 sdev     0.0
seeks:                  1074    (65.6%)
  seek dist (blks):     init  60288,
                        avg   123.6 min      64 max   64000 sdev  1950.9
time to next req(msec): avg  89.512 min   3.135 max 1062.120 sdev 117.073
throughput:             357.4 KB/sec
utilization:            0.08
```

Since lv0hd0 is not a JFS file system, you will not see any references to lv0hd0.

So what files were the most active during our monitoring? We can find that answer in the following output:

```
# awk '/Most Active Files/,/^$/' filemon.out
Most Active Files
------------------------------------------------------------------------
  #MBs  #opns   #rds   #wrs  file                     volume:inode
------------------------------------------------------------------------
 337.3   2059  86358      0  fma.data                 /dev/hd2:342737
 176.7   2057  45244      0  fms.data                 /dev/hd2:342738
  45.6      1   1010    450  rlv0hd0
   9.6      2   2458      0  unix                     /dev/hd2:30988
   6.8     12  66140      0  errlog                   /dev/hd9var:2065
   3.6      2    915      0  htx.errpta               /dev/hd3:71
   1.2      6    318      0  codepoint.cat            /dev/hd9var:2056
   0.8   2059      0   2055  htx.fp3arg               /dev/hd3:57
   0.8   2056      0   2053  htx.fp3arg1              /dev/hd3:62
   0.2      1      0     62  oth.errpt                /dev/hd3:74
   0.2      1     41      0  nasa7.out11              /dev/hd3:4110
   0.2      1     41      0  nasa7.out14              /dev/hd3:4192
   0.2      1     41      0  nasa7.out0               /dev/hd3:4176
   0.2      1     41      0  nasa7.out2               /dev/hd3:4122
   0.2      1     41      0  nasa7.out5               /dev/hd3:4112
   0.2    169      0   2535  htx.fpout15              /dev/hd3:122
   0.1    167      0   2493  htx.fpout1               /dev/hd3:46
   0.1    153      0   2295  htx.fpout8               /dev/hd3:60
   0.1      1      0     35  nasa7.out12              /dev/hd3:4186
   0.1    151      0   2252  htx.fpout12              /dev/hd3:120
```

Let us now find the fma.data file so that we can check it out with the `file place` command (we know that hd2 is the /usr filesystem):

```
# find /usr -name fma.data
/usr/lpp/htx/rules/reg/hxeflp/fma.data
```

We now have both the filename and the path, so we check how the file is allocated on the logical volume:

```
# fileplace -v /usr/lpp/htx/rules/reg/hxeflp/fma.data

File: /usr/lpp/htx/rules/reg/hxeflp/fma.data  Size: 165432 bytes  Vol: /dev/hd2
Blk Size: 4096  Frag Size: 4096  Nfrags: 41   Compress: no
Inode: 342737  Mode: -rwxr-xr-x  Owner: root  Group: system

  Logical Fragment
  ----------------
  0351537-0351544                 8 frags    32768 Bytes,  19.5%
  0351546-0351578                33 frags   135168 Bytes,  80.5%

  41 frags over space of 42 frags:  space efficiency = 97.6%
  2 fragments out of 41 possible:  sequentiality = 97.5%
```

We discover that we have a decent allocation, with approximately 98 percent *space efficiency* and *sequentially*.

To assess the performance effect of file fragmentation, an understanding of how the file is used by the application is required:

- If the application is primarily accessing this file sequentially, the logical fragmentation is more important. At the end of each fragment, read ahead stops. The fragment size is therefore very important.
- If the application is accessing this file randomly, the physical fragmentation is more important. The closer the information is in the file, the less latency there is when accessing.

10.4.1.3 To start with xmperf

The disk I/O statistics can also be monitored with `xmperf`. The required values can be selected in the add values section of `xmperf`. In our case, we simply monitored the I/O transfers to and from the disks, as shown in Figure 28:

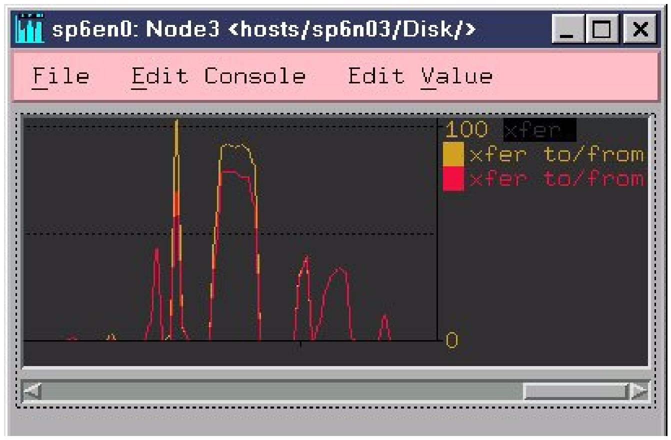

Figure 28. xmperf disk I/O monitor

10.5 Monitoring Network I/O

Monitoring the network I/O usage in a SP node is very important, because an SP by design is a very network intensive system. If the network settings are not set appropriately, it will usually lead to performance degradation, and in some cases, service discontinuance and application malfunctions.

For our purposes here, we assume a TCP/IP network and communication between remote processes based on, mainly, the TCP or UDP protocol. To obtain current settings for different network services and protocols, the following is a shortlist of some of the more important commands to use. See Chapter 4, “Network tuning” on page 21 for more information.

Table 32. Commands for determining network protocol and services settings

Command (option)	Purpose
`no -a`	Configures or displays network attributes.
`nfso -a`	Configures or displays Network File System (NFS) network variables.
`ifconfig`	Configures or displays network interface parameters for a network using TCP/IP.
`netstat`	Shows network status.
`ndb`	Displays network kernel data structures.
`arp -a`	Configures or displays address resolution interfaces.

Command (option)	Purpose
`lsattr`	Displays attribute characteristics and possible values of attributes for devices.

10.5.1 System-wide monitoring

Memory monitoring can be done with many commands and tools. We will try to describe two starting points and how to pursue the resource specifics of memory monitoring.

10.5.1.1 To start with netstat

The `netstat` command is a very versatile network monitoring tool and will show most of what is needed about how the network resources are being used. The following `netstat` command example shows how to start by looking at the traffic overview per network interface (adapter):

```
# netstat -i
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 link#1                          5962     0     6017     0     0
lo0   16896 127         loopback            5962     0     6017     0     0
lo0   16896 ::1                             5962     0     6017     0     0
en0   1500  link#2      0.60.94.e9.4d.b0  113514     0   109486     0     0
en0   1500  192.168.6   sp6n07            113514     0   109486     0     0
tr0   1492  link#3      0.20.35.7a.cb.36       0     0        7     7     0
tr0   1492  192.169.16  sp6tr07                0     0        7     7     0
css0  65520 link#4                        115576     0   102085     0     0
css0  65520 192.168.16  sp6sw07           115576     0   102085     0     0
css1* 65504 link#5                             0     0        0     0     0
css1* 65504 129.10.10   night4s37_00           0     0        0     0     0
```

In the preceding output, we can see that most of the interfaces are up and running, except the last line with css1 that is down (the `*` after the interface name). We can also see all the MTU sizes for each interface and the networks with the primary interface IP address (aliases are not shown). The last columns in the output shows received packets (Ipkts), sent packets (Opkts) and packets with errors that was discarded (Ierrs/Oerrs).

Device drivers

To find out if there are any buffer or allocation problems that could cause delays and interruptions in the traffic flow, check the interface output:

```
# netstat -v|egrep 'STATISTICS|Overflow'
ETHERNET STATISTICS (ent0) :
S/W Transmit Queue Overflow: 0
ETHERNET STATISTICS (ent1) :
S/W Transmit Queue Overflow: 0
TOKEN-RING STATISTICS (tok0) :
S/W Transmit Queue Overflow: 0
```

Note that the Switch adapter (css#) will not show up using `netstat`, so we use a separate command:

```
# estat css0|egrep 'STATISTICS|Overflow'
CSS STATISTICS (css0) :
S/W Transmit Queue Overflow: 0
```

We implement a script (calling it nsi.all) to show all the current available interfaces. But before you run the script, make sure you include `estat` (/usr/lpp/ssp/css/estat) as part of the default path.

```
#!/bin/ksh

netstat -i|
  awk 'NR>1&&$1!~/^lo/{print $1,substr($1,length($1))}'|
    sort -u|
    while read i n;do
      case $i in
        e*) entstat -d "en$n";;
        t*) tokstat -d "tr$n";;
        f*) fddistat -d "fddi$n";;
        a*) atmstat -d "atm$n";;
        c*) estat -d "css$n";;
      esac
    done
```

When we run the script, it will show all interfaces that are Available: and active (that is, not marked with an `*`):

```
# nsi.all|egrep 'STAT|Over'
ETHERNET STATISTICS (ent0) :
S/W Transmit Queue Overflow: 0
DMA Underrun: 0                              DMA Overrun: 0
TOKEN-RING STATISTICS (tok0) :
S/W Transmit Queue Overflow: 0
Receive Overruns : 0
CSS STATISTICS (css0) :
S/W Transmit Queue Overflow: 0
```

Mbufs

Continuing on our mission to examine the network usage, we check the mbuf usage with the -m option to `netstat` (extendednetstats must be set to 1, otherwise `netstat` will not be able to obtain the output we desire):

```
# [[ $(no -o extendednetstats|awk '{print $3}') -gt 0 ]] && netstat -m|head -5
4147 mbufs in use:
1036 Kbytes allocated to mbufs
106 requests for mbufs denied
0 calls to protocol drain routines
0 sockets not created because sockthresh was reached
```

The preceding output, shows that we have had requests for mbufs denied; a more detailed output can be found by looking at the mbuf allocation sizes:

```
# netstat -m|awk '/CPU/||/By size/;$4~/[0-9].*/&&$4>0{print $0}'
******* CPU 0 *******
By size       inuse     calls failed   delayed    free   hiwat   freed
******* CPU 1 *******
By size       inuse     calls failed   delayed    free   hiwat   freed
256            1799   2566588     12         0    1369    1392       1
2048           2048      2807    106         0       2     362       0
******* CPU 2 *******
By size       inuse     calls failed   delayed    free   hiwat   freed
******* CPU 3 *******
By size       inuse     calls failed   delayed    free   hiwat   freed
kernel table      2        16      0       0    1024    2048     0
mcast opts        0         1      0       0       0     128     0
```

We might want to return to examining the different protocol statistics; we can use the -s option to `netstat` to accomplish this task. First, we examine the IP protocol layer by examining all non-zero fields:

```
# netstat -ssp ip
ip:
        23851859 total packets received
        91596 fragments received
        45796 packets reassembled ok
        23804803 packets for this host
        1256 packets for unknown/unsupported protocol
        10 packets forwarded
        23972857 packets sent from this host
        45796 output datagrams fragmented
        91596 fragments created
        1914 successful path MTU discovery cycles
        1429 path MTU rediscovery cycles attempted
        2744 path MTU discovery no-response estimates
        2814 path MTU discovery response timeouts
        179 path MTU discovery decreases detected
        9663 path MTU discovery packets sent
```

Routing

There has been a lot of MTU traffic, so we check the *MTU discovery* settings with the `no` command:

```
# no -a |grep pmtu
         pmtu_default_age = 10
 pmtu_rediscover_interval = 30
        udp_pmtu_discover = 1
        tcp_pmtu_discover = 1
```

The MTU discovery is turned on; an extract from the routing table follows:

```
# netstat -rn
Routing tables
Destination      Gateway           Flags   Refs     Use  If   PMTU  Exp  Groups

Route Tree for Protocol Family 2 (Internet):
default          193.168.207.50    UGc       0        0  en0     -   -
9/8              9.15.193.222      UGc       0        0 css0     -   -
9.15.80.14       9.15.193.222      UGHW      1      523 css0  1500   -
9.15.209.81      9.15.193.222      UGHW      3       96 css0  1500   -
9.223.6.48       9.15.193.222      UGHW      1      232 css0  1492   -
9.223.6.117      9.15.193.222      UGHW      1     3569 css0  1470   -
9.223.6.153      9.15.193.222      UGHW      3     1530 css0  1470   -
127/8            127.0.0.1         U        23   499542  lo0     -   -
146.100.222.190  9.15.193.222      UGHW      2    12606 css0  1500   -
146.100.222.193  9.15.193.222      UGHW      2      369 css0  1500   -
146.100.222.194  9.15.193.222      UGHW      4    15088 css0  1500   -
172.16/16        9.15.193.222      UGc       0        0 css0     -   -
172.25.1.189     193.168.207.50    UGHW      1        6  en0     -   -
172.25.1.193     193.168.207.50    UGHW      1        6  en0     -   -
193.168.206/26   193.168.206.60    U         1  3167277 css0     -   -
193.168.207/26   193.168.207.60    U       631 19033262  en0     -   -
<<< lines omitted >>>
```

This node is connected to the production network through the SP Switch Router, which is why most of the routes are shown for css0. The two lines with MTU of 1470 stands out and might be a cause for additional examination, but since the amount of traffic is small, it does not have any significant performance impact.

Protocols

To check how many packets that have been transferred per protocol, we use the -D option to `netstat`:

```
# netstat -D|egrep '^Source|^IP|^TCP|^UDP'
Source                       Ipkts             Opkts     Idrops     Odrops
IP                          288910            286790        666        100
TCP                         140400            117628          0          0
UDP                         146311            126109        948          0
```

We continue by examining the transport protocol layers UDP and TCP. The following shows the UDP statistics:

```
# netstat -ssp udp
udp:
        3725363 datagrams received
        326586 dropped due to no socket
        34 broadcast/multicast datagrams dropped due to no socket
        3398743 delivered
        3654984 datagrams output
```

There has been a lot of dropped packets, but no socket buffer overflows (the field socket buffer overflows would have been shown with a non-zero value).

To quickly check if we have had any IP queue overflows, use the following command:

```
# netstat -sp ip|grep ipintrq
         0 ipintrq overflows
```

The major part would be examining the TCP statistics, as shown in the following:

```
# netstat -ssp tcp
tcp:
        19973667 packets sent
                18561251 data packets (201385108 bytes)
                5046 data packets (4884008 bytes) retransmitted
                244828 ack-only packets (229111 delayed)
                95 window probe packets
                1157152 window update packets
                4435 control packets
        19743451 packets received
                16985922 acks (for 202685820 bytes)
                14710 duplicate acks
                18330209 packets (750372102 bytes) received in-sequence
                3478 completely duplicate packets (670591 bytes)
                1869 packets with some dup. data (7770 bytes duped)
                12415 out-of-order packets (378482004 bytes)
                8358 window update packets
                141 packets received after close
                141 packets received after close
                1 discarded for bad checksum
        1172 connection requests
        2140 connection accepts
        3158 connections established (including accepts)
        12505 connections closed (including 837 drops)
        100 embryonic connections dropped
        16603768 segments updated rtt (of 16557447 attempts)
        4 resends due to path MTU discovery
        72 path MTU discovery terminations due to retransmits
        1757 retransmit timeouts
                12 connections dropped by rexmit timeout
        134 persist timeouts
        4243 keepalive timeouts
                2965 keepalive probes sent
                307 connections dropped by keepalive
        1 connections in timewait reused
```

We should also check ICMP statistics, as shown in the following:

```
# netstat -sp icmp
icmp:
        326645 calls to icmp_error
        0 errors not generated because old message was icmp
        Output histogram:
                echo reply: 3123
                destination unreachable: 326643
        846 messages with bad code fields
        0 messages < minimum length
        0 bad checksums
        0 messages with bad length
        Input histogram:
                echo reply: 2225
                destination unreachable: 331300
                routing redirect: 2
                echo: 3131
                time exceeded: 17
        3123 message responses generated
```

DNS

We also need to check if the DNS is working properly. The following example will test if the name server is responding to ECHO_REQUEST from `ping`:

```
# ping -c 3 $(awk '/nameserver/{print $2}'  /etc/resolv.conf)
PING 9.12.0.30: (9.12.0.30): 56 data bytes
64 bytes from 9.12.0.30: icmp_seq=0 ttl=255 time=1 ms
64 bytes from 9.12.0.30: icmp_seq=1 ttl=255 time=1 ms
64 bytes from 9.12.0.30: icmp_seq=2 ttl=255 time=1 ms

----9.12.0.30 PING Statistics----
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1/1/1 ms
```

Usually, we just leap ahead and try to resolve a name, as shown here with the `host` command on our own hostname:

```
# host -n $(hostname -s)
sp6en0.msc.itso.ibm.com has address 192.168.6.160
```

Alternatively, we could use the `nslookup` command:

```
nslookup $(hostname -s)
Server:  riscserver.itso.ibm.com
Address:  9.12.0.30

Non-authoritative answer:
Name:    sp6en0.msc.itso.ibm.com
Address:  192.168.6.160
```

To check if the default name resolution order is used or not, use the following command:

```
# print $NSORDER

# grep NSORDER /etc/environment

# cat /etc/netsvc.conf
hosts=local,bind
```

The /etc/netsvc.conf file existed and the order is specified. Reverse the default order by checking the /etc/hosts file before querying the DNS server specified in /etc/resolv.conf.

ARP

To see if we have any problems with the ARP cache, check the number of entries:

```
# arp -an|wc -l
     109
```

Since we have more than 100 ARP entries, we should check the ARP settings with the `no` command:

```
# no -a|grep arp
              arptab_bsiz = 7
                arptab_nb = 25
                 arpqsize = 1
              arpt_killc = 20
```

The system has only the default settings, which is not enough (see Chapter 4, "Network tuning" on page 21**)**. We can monitor the ARP cache by either using `iptrace`, `tcpdump` or a simple script. First we monitor with `tcpdump`, for one hour:

```
# tcpdump -NIi en0 arp >/tmp/tcpdump.en0 & sleep 3600;kill -1 %%
[2]     22728
tcpdump: listening on en0

# tcpdump -NIi tr0 arp >/tmp/tcpdump.tr0 & sleep 3600;kill -1 %%
[2]     22744
tcpdump: (tr0):Promiscuous mode not supported for this interface.
        Continuing ...
tcpdump: listening on tr0
```

Then we just check how many requests our local node has done during the trace period, with the following command:

```
# grep -c $(hostname -s) /tmp/tcpdump.en0
11
```

Even though it is not much at all, we need to check the proximity over time that these requests has occurred; we need to look at the actual trace line from `tcpdump`:

```
# grep $(hostname -s) /tmp/tcpdump.en0
14:55:09.403502336 arp who-has sp6n07 tell sp6en0
14:57:17.223626496 arp who-has sp6n05 tell sp6en0
14:57:22.222190848 arp who-has sp6n03 tell sp6en0
15:01:23.178490880 arp who-has sp6n12 tell sp6en0
15:12:24.221085952 arp who-has sp6n14 tell sp6en0
15:17:39.140759040 arp who-has sp6n03 tell sp6en0
15:18:58.794184192 arp who-has sp6n10 tell sp6en0
15:20:25.895874816 arp who-has sp6n11 tell sp6en0
15:22:21.162817024 arp who-has sp6n09 tell sp6en0
15:22:21.210290688 arp who-has sp6n13 tell sp6en0
15:33:23.757966080 arp who-has sp6n14 tell sp6en0
```

We can use a simple script to monitor the changes occurring in the ARP cache, first by simply checking the number of each network type (if we have multiple network types, it is easy to spot if there are large shifts in the number of types that occupies the cache):

```
#!/bin/ksh
t1=/tmp/arp.1
t2=/tmp/arp.2
trap 'rm -f $t1 $t2' EXIT

printf "%12s %12s %12s %12s %12s\n" "# Deleted" "# Added" "Token-Ring" "Ethernet" "FDDI

while :;do
 arp -an >$t1
 sleep ${1:-3}
 arp -an >$t2
 t=$(grep -c token $t2)
 e=$(grep -c ether $t2)
 f=$(grep -c fddi $t2)
 diff $t1 $t2|grep -c '\<'|read _del
 diff $t1 $t2|grep -c '\>'|read _add
 printf "%12s %12s %12s %12s %12s\n" $_del $_add $t $e $f
done
```

As can be seen below, the ARP cache is too small:

```
# arpmon 1
   # Deleted      # Added   Token-Ring     Ethernet         FDDI
          14           14           89           27            0
          13           13          103           13            0
          14           14           93           23            0
           7            7           85           31            0
           0            0           85           31            0
           0            0           85           31            0
           2            0           83           31            0
          15            0           70           27            0
          16            0           57           24            0
          15            0           45           19            0
          18            0           29           16            0
          15            0           17           12            0
           9            0            7           10            0
           0            0            7           10            0
^C
```

The first part that is highlighted shows *ARP cache thrashing*, the second part shows a full APR cache but with no thrashing, the third part (that is also highlighted) shows a purge of the cache, and the fourth part shows a normal small system cache that is not thrashing.

If we want to check the ARP entries to see if there are any strange values, run the `arp` command, as follows:

```
# # arp -a
  night4n60.ppd.pok.ibm.com (129.40.35.115) at 0:60:94:e9:49:d9 [ethernet]
  night4n62.ppd.pok.ibm.com (129.40.35.116) at 0:60:94:e9:4a:b6 [ethernet]
  night4n05.ppd.pok.ibm.com (129.40.35.70) at 0:4:ac:b1:4b:b5 [ethernet]
  18gate.ppd.pok.ibm.com (9.114.18.254) at 0:0:45:0:df:ab [token ring]
  night4n33.ppd.pok.ibm.com (129.40.35.98) at 0:4:ac:ec:7:98 [ethernet]
  night4n09.ppd.pok.ibm.com (129.40.35.74) at 0:4:ac:ec:b:32 [ethernet]
  orlando.ppd.pok.ibm.com (9.114.18.8) at 0:20:35:7c:d:2e [token ring]
  night4n37.ppd.pok.ibm.com (129.40.35.102) at 0:4:ac:ec:8:d0 [ethernet]
  night4n13.ppd.pok.ibm.com (129.40.35.78) at 0:4:ac:ec:5:44 [ethernet]
  night4n41.ppd.pok.ibm.com (129.40.35.106) at 0:4:ac:ec:1:97 [ethernet]
  night4n17.ppd.pok.ibm.com (129.40.35.82) at 0:4:ac:ec:7:36 [ethernet]
```

This system has both Ethernet and Token-Ring entries, but the cache is not over committed at this point.

10.5.1.2 To start with netpmon

The `netpmon` command is a very powerful low level network monitoring tool. To run `netpmon` for our purposes here, use the device driver tracing, as shown here:

```
# netpmon -o netpmon.out -O dd -tv && sleep 60;trcstop

Enter the "trcstop" command to complete filemon processing

[netpmon: Reporting started]

[netpmon: Reporting completed]

[netpmon: 60.583 secs in measured interval]
```

The example shows how to start by looking at the traffic overview per network interface (adapter):

```
# awk '/^Network.*by Device/,/^\=/' netpmon.out|sed '/^$/d;/^=/d'
Network Device-Driver Statistics (by Device):
--------------------------------------------
                        ----------- Xmit -----------   -------- Recv ---------
Device                   Pkts/s  Bytes/s  Util  QLen   Pkts/s  Bytes/s   Demux
------------------------------------------------------------------------------
token ring 0               0.54      650  0.1% 0.001     1.12       91  0.0002
ethernet 0                 1.58      188  0.0% 0.000     1.65      196  0.0002
```

As this report shows, the networks that this system is connected to are barely utilized by it (the Util column). But we also want to know the statistics for the communication with other nodes and systems:

```
# awk '/^Network.*by Destination/,/^\=/' netpmon.out|sed '/^$/d;/^=/d'
Network Device-Driver Transmit Statistics (by Destination Host):
----------------------------------------------------------------
Host                     Pkts/s  Bytes/s
----------------------------------------
9.12.0.119                 0.50      647
sp6n14                     1.51      178
sp6n01                     0.05        9
9.12.0.114                 0.03        3
sp6n11                     0.02        1
```

So most of our traffic was to sp6n14, as shown in the preceding output.

10.5.1.3 To start with xmperf

The network I/O statistics can also be monitored with `xmperf`, as shown in Figure 29 on page 242. The required values can be selected in the add values section of `xmperf`.

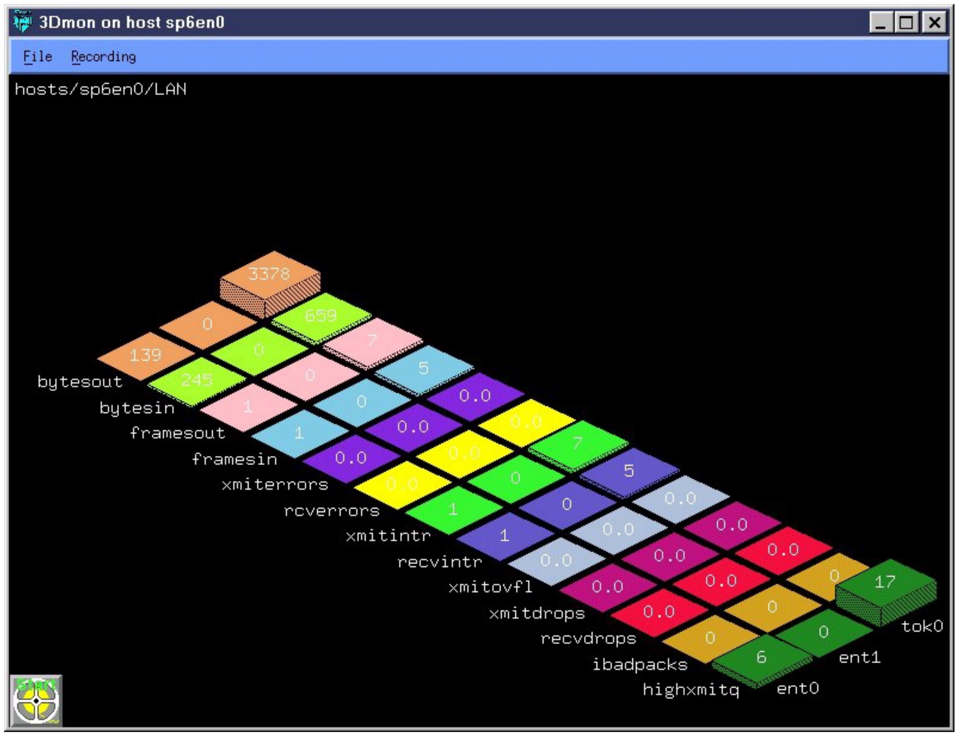

Figure 29. xmperf ntework I/O monitor

Chapter 11. Tools

In the previous chapter, we discussed how resources can be monitored by different commands and tools. In this chapter, we will discuss the different tools in more detail. The emphasis here is on the tool itself and how to use it and what it will tell us when we are using it. Both IBM and non-IBM commands and tools are described. We have also tried to show the benefits of using shell programming to personalize the output in some cases.

11.1 IBM AIX Monitoring tools and commands

AIX provides several monitoring tools to determine performance bottlenecks and to tune the system. Here are most of the important commands/tools used in this book.

11.1.1 acctcom

The `acctcom` command displays information on processes that are in a specified accounting file. To activate process accounting so that the `acctcom` command can be used, create a separate accounting filesystem and then activate the process accounting:

```
# touch /accounting/acctfile
# /usr/sbin/acct/accton /accounting/acctfile
```

To turn off process accounting, execute the `accton` command without any parameter:

```
# /usr/sbin/acct/accton
```

Note

Do not save accounting data in a filesystem that is vital to AIX or applications. Create a separate filesystem for accounting data. The amount of data can be voluminous.

The following is a sample of the `acctcom` output:

```
# acctcom /accounting/acctfile
COMMAND                          START    END          REAL      CPU        MEAN
NAME       USER      TTYNAME  TIME     TIME        (SECS)    (SECS)      SIZE(K)
#accton    root      pts/0    17:53:01 17:53:01      0.02      0.00        0.00
#ls        root      pts/0    17:53:14 17:53:14      0.00      0.00        0.00
#ls        root      pts/0    17:53:16 17:53:16      0.00      0.00      408.00
#ps        root      pts/0    17:53:18 17:53:18      0.08      0.09       28.00
#ping      root      pts/1    19:05:42 17:53:10  82048.00      9.83       40.00
#ls        root      pts/0    17:53:35 17:53:35      0.00      0.00      328.00
#acctcom   root      pts/0    17:53:48 17:53:48      0.05      0.05      116.00
sendmail   root      ?        17:53:50 17:53:50      0.00      0.00      648.00
<<< lines omitted >>>
```

11.1.2 bf/bfrpt

bf (BigFoot) monitors memory usage of applications. bf can be run using executable programs without recompilation and can provide memory footprints for processes, system components, and subroutines. It is possible for bf to identify page sharing between processes and subroutines within the processes. Since the BigFoot functionality operates at the page-level and not at the subroutine-level, if the text segments of two subroutines, or portions of the text segments, reside within a common page, then it is possible for one of the routines to be charged for page references that both make while the other is not charged for those references.

Note

bf can only monitor processes and not threads.

bfrpt is the post-processing utility that can produce graphical and tabular reports.

The following is an example of running bf; we let it run the sleep command for 30 seconds, because it is necessary to specify a command to monitor:

```
# print y|bf -x sleep 30
bf: Load kernel extension bf_kex

Warning:
bf cannot monitor threaded applications, it may crash
the system or give unpredictable results when the
monitored program is threaded application
If the application is threaded, do not continue.

Continue y/[n]i=5

Turning ON BigFoot now.

Buffer Size in Bytes = 520000
Forking Command:  sleep 30



Turning OFF BigFoot now. Copying data to __bf.rpt



bf: A buffer overflow occurred while tracing.  You may want
        to try again with a larger buffer size (-b option)
```

After `bf` has been run, use the `bfrpt -r global` to check the footprint of processes, or use a simpler script like the following to examine the total pages referenced by processes, as reported by `bf` (calling it `bfrpt2`):

```
#!/bin/ksh

/usr/lib/perf/pp ${1:-__bf.rpt} || exit $?

trap '
rm -f __AllByPname __AllByProc __UniqByProc __UniqByProcSort __UniqBySid
rm -f __ftr1 __ftr2 __ftr3 __hdr1 __history.pg __pid.hdr1 __pid.hdr2
' EXIT INTR TERM QUIT

print "Number of Unique Pages Referenced by <Process Name, PID>"
cat __pid.hdr2
sort +2 -3 -nr __UniqByProc
cat __ftr3
```

Running the preceding script `bfrpt2` could give the following result:

```
# bfrpt2 __bf.rpt
Number of Unique Pages Referenced by <Process Name, PID>
     PNAME (20 chars)    PID  TOTAL

             hatsd  14192     92
            dtfile  40242     89
            dtfile  34452     58
               ksh  26136     50
           hardmon   5414     50
             xntpd  22326     42
              cron   9930     19
               gil   1032      1

                    TOTALS    401

                    SHARED     87
```

The preceding output lets us know the total number of unique page references that each process has made during the time `bf` was monitoring.

Important

Be very careful when increasing the buffersize used by `bf`, because setting it wrong might crash the system. Remember that the BigFoot functionality is activated in the VMM and `bf` is not using a normal snapshot method. If you are not sure, leave the buffersize at the default value.

11.1.3 crash

The `crash` command is an interactive utility for examining an operating system image or the running kernel. The `crash` command facility interprets and formats control structures in the system and certain miscellaneous functions for examining a dump. By default, `crash` opens the /dev/mem file and the /unix file so you can run the `crash` command with no arguments to examine an active system.

There are so many uses for `crash`, and we will show some examples.

The following example uses `crash` as a primitive `ps` command to list all nfsd processes in the system process table:

```
# print "th|grep nfsd"|crash 2>/dev/null
 44 s    2c6b     1844  unbound   other  3c   0 0036c7e8      nfsd
 45 s    2d69     1844  unbound   other  3c   0 500014d8      nfsd
 48 z    309f     1844  unbound   other  3c   0               nfsd
 88 s    58fb     1844  unbound   other  3c   0 50028708      nfsd
102 s    669d     1844  unbound   other  3c   1 50028708      nfsd
104 z    6863     1844  unbound   other  3c   0               nfsd
105 s    697f     1844  unbound   other  3d   2 50028708      nfsd
<<< lines omitted >>>
115 z    73ed     1844  unbound   other  3c   0               nfsd
```

As can be seen in the preceding example, we get all the nfsd processes, but we only want to see the nfsd processes that are *not* zombies (z in the second column from the left):

```
# print "th|grep nfsd|grep -v z"|crash 2>/dev/null
 44 s    2c6b     1844  unbound   other  3c   0 0036c7e8      nfsd
 45 s    2d69     1844  unbound   other  3c   0 500014d8      nfsd
 88 s    58fb     1844  unbound   other  3c   0 50028708      nfsd
102 s    669d     1844  unbound   other  3c   0 50028708      nfsd
105 s    697f     1844  unbound   other  3c   0 50028708      nfsd
106 s    6a1d     1844  unbound   other  3c   1 50028708      nfsd
<<< lines omitted >>>
```

Create the following script (calling it `crashmon`) and run it as, for example, `crashmon 1 60`; it will run `crash` each second for a minute:

```
#!/bin/ksh

integer sleep=${1:-5}
integer count=${2:-0}

integer i=0

while :;do
  print 'th -r'|crash
  ((i=$i+1))
  (($count > 0 && $i>=$count)) && break
  sleep $sleep
done
```

11.1.4 col_dump

The col_dump command will access the registers and buffer areas for the SP Switch2 Communications Adapter. To extract some values and monitor what changes by usage, run it as follows:

```
# col_dump -a 0 -g -s -d|egrep -v ':|\.|=|\[ '|sed '/^$/d'|grep '\[[1-9].*\]'
                         parm 0x00000001 [1]
                       status 0x00000002 [2]
                send_cmd_size 0x00000100 [256]
                 send_cmd_cnt 0x00000080 [128]
          send_fifo_entry_size 0x00000200 [512]
         send_fifo_entry_count 0x00000080 [128]
          recv_fifo_entry_size 0x00000200 [512]
         recv_fifo_entry_count 0x00000080 [128]
                  max_windows 0x00000013 [19]
                      version 0x00071703 [464643]
                window_status 0x00000002 [2]
                      node_id 0x00000007 [7]
             recv_hdr_int_ctr 0x28fa2aca [687483594]
            DMA_to_NBA_sw_ctr 0x00000002 [2]
              NBA_push_sw_ctr 0x00000002 [2]
     dump_request_serviced_ctr 0x00000001 [1]
RAMBUS Slot for bit_bucket() and bad_packet() 0x08010000 [134283264]
                           CR 0x22000288 [570425992]
                         HID0 0x3001c024 [805421092]
                         SRR1 0x0000b070 [45168]
```

11.1.5 entstat/tokstat/atmstat/fddistat

These `stat` commands display the statistics gathered by the specified device drivers. The user can optionally specify that the device-specific statistics be displayed in addition to the device generic statistics. If no flags are specified, only the device generic statistics are displayed. This command is also invoked when the `netstat` command is run with the -v flag. The `netstat` command does not issue any `entstat` command flags.

Note

Before doing continous monitoring of the network resource usage with the `stat` commands, reset the counters with the -r option.

To run the `stat` commands, use the -d options, as follows:

```
# entstat -d ent0
# tokstat -d tok0
```

The following shows the part of the device driver statistics that concerns the device drivers mbuf usage:

```
# entstat -dt ent0 |grep -p General
General Statistics:
-------------------
No mbuf Errors: 0
Adapter Reset Count: 0
Driver Flags: Up Broadcast Running
        Simplex AlternateAddress 64BitSupport
        PrivateSegment
```

The adapter device drivers buffer queue (list pointing to mbufs) will show as Max Packets, which is the maximum number of packets that has been on the queue at the same time:

```
<<< lines omitted >>>
Transmit Errors: 0                          Receive Errors: 0
Packets Dropped: 1                          Packets Dropped: 0
                                            Bad Packets: 0
Max Packets on S/W Transmit Queue: 62
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 1
<<< lines omitted >>>
```

Obviously, Receive/Transmit Errors are important, as are Bad/Dropped Packets. Queue Overflow is, of course, how many times there have been requests to transmit packets that could not be performed because the adapters queue was full already. And finally, there is the Current field, which is self explanatory.

The next sample shows the number of collisions for the Ethernet network:

```
# entstat -d ent0|grep -p "Broadcast Packets"
Broadcast Packets: 1231                     Broadcast Packets: 18218
Multicast Packets: 2                        Multicast Packets: 14
No Carrier Sense: 0                         CRC Errors: 0
DMA Underrun: 0                             DMA Overrun: 0
Lost CTS Errors: 0                          Alignment Errors: 0
Max Collision Errors: 0                     No Resource Errors: 0
Late Collision Errors: 0                    Receive Collision Errors: 0
Deferred: 6422                              Packet Too Short Errors: 0
SQE Test: 0                                 Packet Too Long Errors: 0
Timeout Errors: 0                           Packets Discarded by Adapter: 0
Single Collision Count: 3566                Receiver Start Count: 0
Multiple Collision Count: 2371
Current HW Transmit Queue Length: 1
```

An excessive amount of collisions will have a degrading effect on network performance. To reduce the number of collisions on an Ethernet network, a different network topology must be used (for example, different switches creating vlans, changing broadcast options, and so on).

The next sample is a portion of the detailed output for an Ethernet adapter:

```
# entstat -dt ent0 |grep -p "Adapter Specific Statistics"
IBM 10/100 Mbps Ethernet PCI Adapter Specific Statistics:
------------------------------------------------
Chip Version: 26
RJ45 Port Link Status : up
Media Speed Selected: 100 Mbps Full Duplex
Media Speed Running: 100 Mbps Full Duplex
Receive Pool Buffer Size: 2048
Free Receive Pool Buffers: 1791
No Receive Pool Buffer Errors: 0
Inter Packet Gap: 96
Adapter Restarts due to IOCTL commands: 37
Packets with Transmit collisions:
 1 collisions: 3566       6 collisions: 0         11 collisions: 0
 2 collisions: 1828       7 collisions: 0         12 collisions: 0
 3 collisions: 479        8 collisions: 0         13 collisions: 0
 4 collisions: 60         9 collisions: 0         14 collisions: 0
 5 collisions: 4         10 collisions: 0         15 collisions: 0
Excessive deferral errors: 0x0
```

For Ethernet, it is very interesting to look at the number of transfer collision levels. At the end of the listing, there is a Packets with Transmit collisions table. The first position is the same as the field Single Collision Count and the rest are 2 to 16 number of times the collision detection backoff has been performed for sending packets (a summary is in the Multiple Collision Count field). Since the time increases for each additional packet transmission, network traffic is degraded if multiple collisions occur for the same packet and this is occurring frequently.

As for Token-Ring, since it is a more complicated protocol than Ethernet, it has more sophisticated performance degrading possibilities. The following is a normal output from the General section of the `tokstat` report:

```
# tokstat -d tok0|grep -p General
General Statistics:
-------------------
No mbuf Errors: 0                              Lobe Wire Faults: 0
Abort Errors: 0                                AC Errors: 0
Burst Errors: 11                               Frame Copy Errors: 0
Frequency Errors: 0                            Hard Errors: 0
Internal Errors: 0                             Line Errors: 1
Lost Frame Errors: 1                           Only Station: 0
Token Errors: 0                                Remove Received: 0
Ring Recovered: 0                              Signal Loss Errors: 0
Soft Errors: 3                                 Transmit Beacon Errors: 0
Driver Flags: Up Broadcast Running
        AlternateAddress 64BitSupport ReceiveFunctionalAddr
        16 Mbps
```

Some of many interesting things to notice are the Only Station and Token Errors fields. The Only Station field indicates that the local adapter is not aware of any other adapters up-stream. If there are more nodes on the Token-Ring up-stream from this adapter, it should not be zero.

The Token Errors field indicates the number of times the current token on the ring got lost, if this adapter was the Token-Ring *active monitor*.

If there is a disagreement among the Token-Ring adapters on the network as to which one should be the active monitor, there can be a lot of unnecessary traffic concerning acquiring the active monitor role. Since it is the role of the active monitor to generate new tokens, this malfunctioning state will lead to performance degradation that is hard to detect.

11.1.6 estat

The `estat` command displays the statistics gathered by the switch device drivers (css#). The -d option that can be specified shows the number of open windows. This command is *not* invoked when the `netstat` command is run with the -v flag.

The following shows how to run the command and what to expect from the output:

```
# /usr/lpp/ssp/css/estat -d  css0
-------------------------------------------------------------
CSS STATISTICS (css0) :
Elapsed Time: 5 days 18 hours 48 minutes 45 seconds
Transmit Statistics:                          Receive Statistics:
--------------------                          -------------------
Packets: 7091171                              Packets: 6927492
Bytes: 924133755                              Bytes: 414985141
Interrupts: 0                                 Interrupts: 0
Transmit Errors: 0                            Receive Errors: 0
Packets Dropped: 0                            Packets Dropped: 0
Max Packets on S/W Transmit Queue: 0          Bad Packets: 0
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 0
Broadcast Packets: 0                          Broadcast Packets: 0
General Statistics:
-------------------
No mbuf Errors: 0
High Performance Switch Specific Statistics:
----------------------------------------------------------
Windows open: 5
```

11.1.7 filemon

`filemon` monitors a trace of file system and I/O system events, and reports performance statistics for files, virtual memory segments, logical volumes, and physical volumes. `filemon` is useful to those whose applications are believed to be disk-bound, and want to know where and why.

The filemon command uses the AIX system trace facility. Currently, the trace facility only supports one output stream. Consequently, only one trace process can be active at a time. If another trace process is already running, the filemon command will respond with an error message.

The following example shows how to run filemon. To stop filemon tracing of I/O, trcstop must be issued, and it is when this is done that filemon writes the output. To have filemon monitor I/O during a time interval, just run the sleep program with the specified amount of seconds and then the trcstop program:

```
# filemon -o filemon.out -O all && sleep 60;trcstop

Enter the "trcstop" command to complete filemon processing

[filemon: Reporting started]
[filemon: Reporting completed]

[filemon: 96.371 secs in measured interval]
```

The output from filemon can be quite extensive, and to quickly find out if something is in need of attention, we filter it with the awk command, as in most of our examples below, to extract specific summary tables from the filemon output file. We have used the all option and then we will use the awk command to extract relevant parts, but we could have used any of these options:

lf	Logical file level
vm	Virtual memory level
lv	Logical volume level
pv	Physical volume level
all	Short for lf, vm, lv, pv

When analyzing the output, keep in mind that the avg (average) measurement is an indication of efficiency and that the sdev measurement indicates the level of fragmentation (standard deviation).

Summary table for disk I/O

In the following example, we only extract the summary table for disk I/O:

```
# awk '/Most Active Physical Volumes/,/^$/' filemon.out
Most Active Physical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume              description
------------------------------------------------------------------------
  0.24     16  50383  171.9  /dev/hdisk0         N/A
  0.84 370680 372028 3853.4  /dev/hdisk1         N/A
  0.08  68608  36160  357.4  /dev/hdisk2         N/A
```

The disk with the highest transfer rate and utilization is hdisk1, which is 84 percent utilized at a 3.8 MB transfer rate. The fields are interpreted as follows:

util — Utilization of physical volume. Note: logical volume I/O requests start before and end after physical volume I/O requests. For that reason, total logical volume utilization will appear to be higher than total physical volume utilization.

#rblk — Number of 512-byte blocks read from physical volume.

#wblk — Number of 512-byte blocks written to physical volume.

KB/s — Average transfer data rate in KB per second.

volume — Physical volume name.

description — Simple description of the physical volume type, for example, CD-ROM SCSI, 36.0 GB SCSI disk.

Summary table for file I/O

In the following example, we only extract the summary table for file I/O:

```
# awk '/Most Active Files/,/^$/' filemon.out
Most Active Files
------------------------------------------------------------------------
  #MBs  #opns   #rds   #wrs  file                volume:inode
------------------------------------------------------------------------
 180.8      1  46277      0  iptrace.out         /dev/hd3:17
 180.8      1      0  46277  ipreport.out        /dev/hd3:107
   0.0      2      4      0   ksh.cat            /dev/hd2:12741
   0.0      1      2      0   cmdtrace.cat       /dev/hd2:12610
```

We notice heavy reading of the iptrace.out file and writing of the ipreport.out. The fields are interpreted as follows:

#MBs — Total number of MBs transferred over a measured interval for this file. The rows are sorted by this field in decreasing order.

#opns — Number of opens for files during measurement period.

#rds Number of read calls to file.

#wrs Number of write calls to file.

file File name (the full path name is in detailed report).

volume:inode The logical volume that the file resides in and the inode number of the file in the associated file system. This field can be used to associate a file with its corresponding persistent segment shown in the detailed VM segment reports. This field may be blank for temporary files created and deleted during execution.

Summary table for logical volume I/O

In the following example, we only extract the summary table for logical volume I/O:

```
# awk '/Most Active Logical Volumes/,/^$/' filemon.out

Most Active Logical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume             description
------------------------------------------------------------------------
  1.00 370664 370768 3846.8  /dev/hd3           /tmp
  0.02      0    568    2.9  /dev/hd8           jfslog
  0.01      0    291    1.5  /dev/hd9var        /var Frag_Sz.= 512
  0.00      0    224    1.2  /dev/hd4           /
  0.00      0     25    0.1  /dev/hd1           /home Frag_Sz.= 512
  0.00     16    152    0.9  /dev/hd2           /usr
```

The logical volume hd3 with filesystem /tmp is fully utilized at 3.8 MB transfer rate. The fields are interpreted as follows:

util Utilization of logical volume.

#rblk Number of 512 byte blocks read from logical volume.

#wblk Number of 512 byte blocks written to logical volume.

KB/s Average transfer data rate in KB per second.

volume Logical volume name.

description Either the file system mount point or the LV type (paging, jfslog, boot, or sysdump). For example, the LV /dev/hd2 is /usr and /dev/hd8 is jfslog. There may also be the word compressed; this means all data is compressed automatically using LZ compression before being written to disk, and all data is uncompressed automatically when read from disk. The Frag_Sz shows the fragment size; 512 means that there can be four fragments in one page.

To check the details of the highest utilized logical volumes, create a script as shown (calling it filemon.lvdetail) and then run it using the `filemon` output file as input:

```
#!/bin/ksh

file=${1:-filemon.out}
switch=${2:-0.20}       # 20%

# extract the summary table...
awk '/Most Active Logical Volumes/,/^$/' $file|
# select logical volumes starting from line 5 and no empty lines...
  awk 'NR>4&&$0!~/^$/{if ($1 >= switch)print $5}' switch=$switch|
    while read lv;do
# strip the /dev/ stuff and select the detail section.
      awk '/VOLUME: \/dev\/'${lv##*/}'/,/^$/' $file
    done
```

For our continuing example, it would result in the following output:

```
# filemon.lvdetail filemon.out 0.1
VOLUME: /dev/hd3  description: /tmp
reads:                  6896    (0 errs)
  read sizes (blks):    avg    53.8 min       8 max      64 sdev    17.7
  read times (msec):    avg  31.251 min   0.484 max 680.707 sdev  17.817
  read sequences:       6034
  read seq. lengths:    avg    61.4 min       8 max      72 sdev     9.4
writes:                 11739   (0 errs)
  write sizes (blks):   avg    31.6 min       8 max      32 sdev     2.9
  write times (msec):   avg  26.120 min   4.440 max 671.314 sdev  19.814
  write sequences:      5922
  write seq. lengths:   avg    62.6 min       8 max     128 sdev     7.4
seeks:                  11956   (64.2%)
  seek dist (blks):     init   1160,
                        avg 476295.2 min       8 max 1703560 sdev 386639.7
time to next req(msec): avg   5.171 min   0.001 max 665.182 sdev  11.744
throughput:             3846.8 KB/sec
utilization:            1.00
```

The description for the detailed output from the `filemon` command for logical, and physical volumes are as follows:

reads	Number of read requests made against the volume.
read sizes (blks)	The read transfer-size statistics (avg/min/max/sdev), in units of 512-byte blocks.
read times (msec)	The read response-time statistics (avg/min/max/sdev), in milliseconds.

read sequences	Number of read sequences. A sequence is a string of 512-byte blocks that are read consecutively and indicate the amount of sequential access.
read seq. lengths	Statistics describing the lengths of the read sequences, in blocks.
writes	Number of write requests made against the volume.
write sizes (blks)	The write transfer-size statistics (avg/min/max/sdev), in units of 512-byte blocks.
write times (msec)	The write response-time statistics (avg/min/max/sdev), in milliseconds.
write sequences	Number of write sequences. A sequence is a string of 512-byte blocks that are written consecutively.
write seq. lengths	Statistics describing the lengths of the write sequences, in blocks.
seeks	Number of seeks that preceded a read or write request; also expressed as a percentage of the total reads and writes that required seeks.
seek dist (blks)	Seek distance statistics, in units of 512-byte blocks. In addition to the usual statistics (avg/min/max/sdev), the distance of the initial seek operation (assuming block 0 was the starting position) is reported separately. This seek distance is sometimes very large, so it is reported separately to avoid skewing the other statistics.
seek dist (cyls)	Hard files only; seek distance statistics, in units of disk cylinders.
time to next req	Statistics (avg/min/max/sdev) describing the length of time, in milliseconds, between consecutive read or write requests to the volume. This column indicates the rate at which the volume is being accessed.
throughput	Total volume throughput, in Kilobytes per second.
utilization	Fraction of time the volume was busy. The entries in this report are sorted by this field, in decreasing order.

11.1.8 fileplace

`fileplace` displays the placement of a file's blocks within a file system. Logically contiguous files may be physically fragmented on disk, depending on the available free space at the time the file is created. `fileplace` can be used to examine and assess the efficiency of a file's placement on disk.

This example shows the logical layout for the file ipreport.out:

```
# fileplace -vl ipreport.out

File: ipreport.out  Size: 598162 bytes  Vol: /dev/hd3
Blk Size: 4096  Frag Size: 4096  Nfrags: 147   Compress: no
Inode: 21  Mode: -rw-------  Owner: root  Group: system

  Logical Fragment
  ----------------
  0000286-0000293                 8 frags    32768 Bytes,   5.4%
  0000298-0000301                 4 frags    16384 Bytes,   2.7%
  0000369-0000376                 8 frags    32768 Bytes,   5.4%
  0000390-0000397                 8 frags    32768 Bytes,   5.4%
<<< lines omitted >>>
  0000544-0000551                 8 frags    32768 Bytes,   5.4%
  0000560-0000563                 4 frags    16384 Bytes,   2.7%
  0000567-0000570                 4 frags    16384 Bytes,   2.7%
  0000579-0000586                 8 frags    32768 Bytes,   5.4%
  0000596-0000603                 8 frags    32768 Bytes,   5.4%
  0000620-0000627                 8 frags    32768 Bytes,   5.4%
  0000633-0000640                 8 frags    32768 Bytes,   5.4%
  0000655-0000662                 8 frags    32768 Bytes,   5.4%
  0000671-0000677                 7 frags    28672 Bytes,   4.8%

  147 frags over space of 392 frags:  space efficiency = 37.5%
  20 fragments out of 147 possible:  sequentiality = 87.0%
```

This example shows the physical layout of the file ipreport.out:

```
# fileplace -vp ipreport.out

File: ipreport.out  Size: 598162 bytes  Vol: /dev/hd3
Blk Size: 4096  Frag Size: 4096  Nfrags: 147   Compress: no
Inode: 21  Mode: -rw-------  Owner: root  Group: system

  Physical Addresses (mirror copy 1) Logical Fragment
  ---------------------------------- --------- -------
  2458430-2458437  hdisk1 8 frags    32768 Bytes,   5.4%    0000286-0000293
  2458442-2458445  hdisk1 4 frags    16384 Bytes,   2.7%    0000298-0000301
  2458513-2458520  hdisk1 8 frags    32768 Bytes,   5.4%    0000369-0000376
  2458534-2458541  hdisk1 8 frags    32768 Bytes,   5.4%    0000390-0000397
<<< lines omitted >>>
  2458688-2458695  hdisk1 8 frags    32768 Bytes,   5.4%    0000544-0000551
  2458704-2458707  hdisk1 4 frags    16384 Bytes,   2.7%    0000560-0000563
  2458711-2458714  hdisk1 4 frags    16384 Bytes,   2.7%    0000567-0000570
  2458723-2458730  hdisk1 8 frags    32768 Bytes,   5.4%    0000579-0000586
  2458740-2458747  hdisk1 8 frags    32768 Bytes,   5.4%    0000596-0000603
  2458764-2458771  hdisk1 8 frags    32768 Bytes,   5.4%    0000620-0000627
  2458777-2458784  hdisk1 8 frags    32768 Bytes,   5.4%    0000633-0000640
  2458799-2458806  hdisk1 8 frags    32768 Bytes,   5.4%    0000655-0000662
  2458815-2458821  hdisk1 7 frags    28672 Bytes,   4.8%    0000671-0000677

  147 frags over space of 392 frags:  space efficiency = 37.5%
  20 fragments out of 147 possible:  sequentiality = 87.0%
```

11.1.9 hatstune

The `hatstune` command can be used to tune the High Availability Topology Services (HATS) heart beat functionality. It will update the SDR for the TS_Config object. The default settings follow:

```
# SDRGetObjects TS_Config
Frequency    Sensitivity  Run_FixPri   FixPri_Value Log_Length   Pinning
          1            4            1           38         5000 ""
```

To check the current settings with the `hatstune` command, use the following:

```
# /usr/sbin/rsct/bin/hatstune -v
Current HATS tunable values:
    Frequency: 1
    Sensitivity: 4
    Running fixed priority: 38
    Maximum number of lines in log file: 5000
    Pinning in real memory:
```

The options that we would use to tune the execution of hats are as follows:

Frequency (-f) Controls how often Topology Services sends a heartbeat to its neighbors. The value is interpreted as the number of seconds between heartbeats. On a system with a high amount of paging activity or other heavy workload, this number should be kept as small as possible. The frequency is an integer value of heart beat frequency in seconds. The valid frequency range is [1, 30] or a special keyword “default.” The default frequency value is used if the frequency is “default.”

Sensitivity (-s) Controls the number of missed heartbeat messages that will cause a Death in Family message to be sent to the Group Leader. Heartbeats are not considered missing until it has been twice the interval indicated by the Frequency attribute. The sensitivity is an integer value of heart beat sensitivity. The valid sensitivity range is [4, 40] or a special keyword “default.” The default sensitivity value will be used if sensitivity is the special keyword “default.”

To set the values to default, run `hatstune` in the following way:

```
# /usr/sbin/rsct/bin/hatstune -f default -s default
The following HATS tunables are changed:
    Default heart beat frequency changed to 1.
    Default heart beat sensitivity changed to 4.
```

It is important that the values are not changed without monitoring the effect. If the values are too low, nodes might not have time enough to answer and will be perceived as being off-line. If the values are too high, the nodes might be off-line but are not detected as being off-line. Increasing the settings in small increments while monitoring is the best approach. Usually, it is not necessary to change the priority values (which can also be done). When changing the frequency and sensitivity, start by doubling the values and see if that is enough, as is shown in the following example:

```
# /usr/sbin/rsct/bin/hatstune -f 2 -s 8
The following HATS tunables are changed:
    Default heart beat frequency changed to 2.
    Default heart beat sensitivity changed to 8.
```

You can verify that the changes have taken place in the SDR as shown in the following:

```
# SDRGetObjects TS_Config
Frequency    Sensitivity  Run_FixPri   FixPri_Value Log_Length   Pinning
         2            8            1           38         5000 ""
```

11.1.10 ifcl_dump

The `ifcl_dump` command (new for the SP Switch2 in PSSP 3.2) accesses the pseudo device driver (csspd#) that maintains the nodes connectivity matrix. This will show the currently available connections over the switch that the node knows about. The first example shows a node with no switch connections:

```
# ifcl_dump -m
CM reachable structure
           |    ret_code       0x00000000 [0]
           |    inout          0x00000000 [0]
           |    reachfrom      0x00000000 [0]
         |  nodelist
```

The following example shows a node with two working fault service connections:

```
# ifcl_dump -m
CM reachable structure
           |    ret_code       0x00000000 [0]
           |    inout          0x00000002 [2]
           |    reachfrom      0x00000001 [1]
         |  nodelist
        0x0003  0x0007
```

11.1.11 iostat

`iostat` is foremost a tool to get a first impression of whether a system has an I/O-bound performance problem or not. The tool, however, reports the same CPU activity statistics as `vmstat` does, which is a percentage breakdown of user mode, system mode, idle time, and waits for disk I/O. For multiprocessor systems, the CPU values are global averages among all processors. Also, the I/O wait state is defined system-wide and not per processor.

The following example shows how we monitor the system with `iostat` saving output to a file (it runs for 60 minutes in 5 second intervals, and the sample data is not shown here in its entirety):

```
# iostat -d 5 720|tee /tmp/iostat.out
Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk2           0.0       0.5       0.0       4247      2114
hdisk3           0.0       0.0       0.0          0         0
hdisk4           0.0       0.0       0.0          0         0
hdisk5           0.0       0.0       0.0          0         0
hdisk0           0.1       3.5       0.2      41175      3360
hdisk1           2.0      11.7       2.5      49905    100387
hdisk6           0.0       4.1       0.1       7440     44436
<<< lines omitted >>>
```

To make some summaries, we use two scripts, and the first one is for reporting the most heavily loaded disk:

```
# iostat.max /tmp/iostat.out
Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk6         100.0     16944.0     135.8          0     84720
hdisk6         100.0     16181.5     165.1          4     80944
hdisk6         100.0     16102.4     161.2          4     80508
hdisk6         100.0     16072.8     163.7          4     80400
hdisk6         100.0     16028.8     162.6          4     80140
hdisk6         100.0     15754.4     159.2          4     78768
hdisk6         100.0     15231.2     184.0          8     76148
hdisk6         100.0     14891.2     184.2          8     74448
hdisk6         100.0     14688.7     185.5          8     73472
hdisk6         100.0     14415.2     181.4          8     72068
hdisk6         100.0     14272.8     147.6          4     71360
hdisk6         100.0     13997.8     177.3          8     70016
hdisk6          99.8     16226.3     149.9          4     81168
```

This is the sample script used (called iostat.max); it first goes through the output to find the disk with the highest peak usage and then extracts all records for this disk using half the peak value as a breaking point (stripping off the first summary output from `iostat` as well):

```
#!/bin/ksh

input=${1:-iostat.out}
awk '{
    if (!first[$1]>0) {first[$1]=NR;next}
    if($2>max){max=$2;disk=$1}}END{print disk,max}
' $input|read disk max
print "Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn"
awk '$1~/^'$disk'/&& $2>'$((${max%%.*}/2))'' $input|sort -nrk2
```

To summarize the read and write transfer from and to a disk, a script such as the one that follows (called iostat.maxdata) could be used (stripping off the first summary output from `iostat` as well):

```
#!/bin/ksh

printf "%-10.10s %12s %12s %12s\n" "Disk" "Read(KB)" "Write(KB)" "#samples"
awk '/^hdisk/{
  if (!first[$1]>0) {first[$1]=NR;next}
  disk[$1]++
  write[$1]=write[$1]+$6
  read[$1]=read[$1]+$5
}
END{
    for (i in disk)
        printf "%-10.10s %12d %12d %12d\n",i,read[i],write[i],disk[i]
}' ${1:-iostat.out}|sort -rn -k2 -k3
```

With the same input data as the previous example, it would generate the following output:

```
# iostat -d 5 72 >/tmp/iostat.out
# iostat.maxdata /tmp/iostat.out
Disk           Read(KB)    Write(KB)     #samples
hdisk6             7432      1006340         3599
hdisk0             7364            0         3599
hdisk1                8        14627         3599
hdisk5                0            0         3599
hdisk4                0            0         3599
hdisk3                0            0         3599
hdisk2                0            0         3599
```

11.1.12 iptrace/ipreport/ipfilter

Monitoring the network traffic with `iptrace` (and `tcpdump`) can often be very useful in determining why network performance is not as expected. For example, in reviewing an instance of the ip trace, a pattern was found that indicated nfsd usage. It was discovered that out of 112,000 packets, 42,000 packets were replies that had the error “NFS: Stat: 970) Error- Stale NFS file handle.” Basically, a large amount of traffic (roughly half) was caused by the inability of clients to access an old file handle.

iptrace

The `iptrace` daemon records Internet packets received from configured interfaces. Command flags provide a filter so that the daemon traces only packets meeting specific criteria. Packets are traced only between the local host on which the `iptrace` daemon is invoked and the remote host.

The following example shows how to trace a bi-directional connection between a server host and a client (remotenode), save the output in a file, wait for 30 seconds and then stop the trace:

```
# startsrc -s iptrace -a "-a -b -d remotenode /tmp/iptrace.out" &&
> sleep 30 &&
> stopsrc -s iptrace
```

ipreport

To format the data file generated by `iptrace`, run the `ipreport` command.

The `ipreport` command generates a trace report from the specified trace file created by the `iptrace` command. The following example uses the trace file generated by `iptrace` to generate a readable report, excluding name lookup, and only reporting on the 100 first packets, starting from packet number 55 and including RPC information:

```
# ipreport -c 100 -j 55 -v -N -rs /tmp/iptrace.out >/tmp/ipreport.out
```

The following is an example of the output from `ipreport` after running `ping` between two nodes over Ethernet:

```
ETH: ====( 98 bytes received on interface en0 )==== 10:41:20.509152590
ETH:     [ 00:04:ac:ec:07:98 -> 00:04:ac:ec:08:d0 ]  type 800  (IP)
IP:      < SRC =    129.40.35.98 >
IP:      < DST =   129.40.35.102 >
IP:      ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=63399, ip_off=0
IP:      ip_ttl=255, ip_sum=7ae8, ip_p = 1 (ICMP)
ICMP:    icmp_type=0 (ECHO_REPLY)  icmp_id=16974  icmp_seq=22
```

This is the ECHO_REPLY packet that is returned to the host issuing the ECHO_REQUEST (in this example, it is the one running the `ping` command).

This example shows the initiation of a TCP connection between two nodes (from SRC to DST):

```
ETH: ====( 74 bytes received on interface en0 )==== 12:22:05.191609117
ETH:     [ 00:04:ac:ec:07:98 -> 00:04:ac:ec:08:d0 ]  type 800  (IP)
IP:      < SRC =    129.40.35.98 >
IP:      < DST =   129.40.35.102 >
IP:      ip_v=4, ip_hl=20, ip_tos=0, ip_len=60, ip_id=41626, ip_off=0
IP:      ip_ttl=60, ip_sum=9309, ip_p = 6 (TCP)
TCP:     <source port=34308, destination port=2049(shilp) >
TCP:     th_seq=f47ddc71, th_ack=0
TCP:     th_off=10, flags<SYN>
TCP:     th_win=65535, th_sum=4b0a, th_urp=0
TCP:         mss 1460
TCP:         nop
TCP:         wscale 1
TCP:         nop
<<< lines omitted >>>
```

Note the request for *message segment size* of 1460 (a similar sample for TCP initiation is also shown for the `tcpdump` command; see Section 11.1.32, “tcpdump” on page 317 for more information). This sample is the reply to the initiation request (note the SYN and ACK in the `flags` field):

```
ETH: ====( 74 bytes transmitted on interface en0 )==== 12:22:05.191741778
ETH:     [ 00:04:ac:ec:08:d0 -> 00:04:ac:ec:07:98 ]  type 800  (IP)
IP:      < SRC =   129.40.35.102 >
IP:      < DST =     129.40.35.98 >
IP:      ip_v=4, ip_hl=20, ip_tos=0, ip_len=60, ip_id=51148, ip_off=0
IP:      ip_ttl=60, ip_sum=6dd7, ip_p = 6 (TCP)
TCP:     <source port=2049(shilp), destination port=34308 >
TCP:     th_seq=b29207af, th_ack=f47ddc72
TCP:     th_off=10, flags<SYN | ACK>
TCP:     th_win=59368, th_sum=5530, th_urp=0
TCP:         mss 1460
TCP:         nop
TCP:         wscale 0
TCP:         nop
<<< lines omitted >>>
```

ipfilter

The `ipfilter` sorts the file provided by the `ipreport` command, provided the -r (for NFS reports) and -s (for all reports) flag has been used to generate the report. It provides information about NFS, UDP, TCP, IPX and ICMP headers in table form. Information can be displayed together, or separated by headers into different files. It also provides information about NFS calls and replies.

To run an `ipfilter` script that will generate a specific report for different protocols (header types in the `ipreport` output file), use the following command:

```
# ipfilter -s untxc /tmp/ipreport.out
```

Table 33 shows the different header types and options.

Table 33. Header types

Header Type	Header type option	Output file name
NFS (RPC)	n	ipfilter.nfs
TCP	t	ipfilter.tcp
UDP	u	ipfilter.udp
ICMP	c	ipfilter.icmp
IPX (PC protocol)	x	ipfilter.ipx

If no options are specified, `ipfilter` will generate a file containing all protocols (ipfilter.all).

The following is a sample output taken from ipreport.nfs generated by `ipfilter`; it can be used to get an overview of the NFS traffic:

```
# more ipfilter.nfs
                                 Operation Headers: NFS
                Ports
     ------------------------------------
 pkt.      Time          Source           Dest.   Length Seq #      Ack #
  Source     Destination Net_Interface Operation
--------------------------------------------------------------------------------
---------------------------------------------------
1388 08:43:07.881449     192.168.6.7   192.168.6.160  176, a1441f13,  db32c357
     34299,  2049(shilp)           en0 TCP ACK PUSH NFS
1391 08:43:07.903165   192.168.6.160     192.168.6.7  208, db32c357,  a1441f9b 2
049(shilp),       34299           en0 TCP ACK PUSH NFS
<<< lines omitted >>>
```

The following is sample output taken from ipreport.tcp generated by `ipfilter`, it gives a good overview of the TCP packet flow:

```
# more ipfilter.tcp
                                Operation Headers: TCP
                Ports
     ------------------------------------
 pkt.      Time           Source           Dest.   Length Seq #      Ack #
  Source     Destination Net_Interface Operation
--------------------------------------------------------------------------------
---------------------------------------------------
   1 18:35:06.814647        9.12.0.6      9.12.0.114   63, 2610f9ea,    3c0813
23(telnet),         1195           tr0 TCP ACK PUSH
   2 18:35:06.891148      9.12.0.105        9.12.0.6   40,   93da21,  df807567
      6000,        46398           tr0 TCP ACK
   3 18:35:06.957874        9.12.0.6      9.12.0.105   60, df807567,    93da21
     46398,         6000           tr0 TCP ACK PUSH
<<< lines omitted >>>
```

The following is sample output taken from ipreport.udp generated by `ipfilter`; it shows the UDP packet flow:

```
# more ipfilter.upd
                                Operation Headers: UDP
                Ports
     ------------------------------------
 pkt.      Time           Source           Dest.   Length Seq #      Ack #
  Source     Destination Net_Interface Operation
--------------------------------------------------------------------------------
---------------------------------------------------
   6 18:35:07.116037       9.12.0.15      9.12.0.255  239,                      1
38(netbios- 138(netbios-           tr0 UDP
   7 18:35:07.139761   192.168.6.160    192.168.6.14  104,                      1
0000(hats.s 10000(hats.s           en0 UDP
23 18:35:08.248904    192.168.6.10   192.168.6.160   76,
  123(ntp),     123(ntp)           en0 UDP
<<< lines omitted >>>
```

11.1.13 lsps

The `lsps` command displays the characteristics of paging spaces, such as the paging-space name, physical-volume name, volume-group name, size, percentage of the paging space used, whether the space is active or inactive, and whether the paging space is set to automatic activation at boot (IPL). The following are a couple of examples that shows one paging space on one disk:

```
# lsps -a
Page Space  Physical Volume   Volume Group    Size   %Used  Active  Auto
hd6         hdisk0            rootvg        9216MB       1     yes   yes

# lsps -s
Total Paging Space   Percent Used
      9216MB              1%
```

Since paging is usually to be avoided, use `lsps` (or `svmon` [see Section 11.1.31, “svmon” on page 314) or `pstat` (see Section 11.1.27, “pstat” on page 308]) to monitor paging space usage.

11.1.14 lspv/lslv/lsvg/getlvcb

These commands are useful for determining the logical volume layout and gives a good overview.

lspv

The `lspv` command is useful for displaying information about the physical volume and its logical volume content and logical volume allocation layout, as the following two examples show:

```
# lspv -l hdisk0
hdisk0:
LV NAME              LPs   PPs   DISTRIBUTION          MOUNT POINT
hd5                  1     1     01..00..00..00..00    N/A
hd6                  288   288   00..108..108..72..00 N/A

# lspv -p hdisk0
hdisk0:
PP RANGE  STATE   REGION        LV NAME             TYPE       MOUNT POINT
  1-1     used    outer edge    hd5                 boot       N/A
  2-109   free    outer edge
110-217   used    outer middle  hd6                 paging     N/A
218-325   used    center        hd6                 paging     N/A
326-397   used    inner middle  hd6                 paging     N/A
398-433   free    inner middle
434-542   free    inner edge
```

In the preceding examples, we can see that the hd6 logical volume is nicely placed in the center area of the disk, the distribution being 108 LPs in the very center, 108 LPs in the outer middle, and 72 LPs in the inner middle part of the disk.

lslv

The `lslv` command displays the characteristics and status of the logical volume, as the following example shows:

```
# lslv -l hd6
hd6:N/A
PV                COPIES          IN BAND       DISTRIBUTION
hdisk0            288:000:000     37%           000:108:108:072:000
```

As can be seen above, the `lspv` and `lslv` shows the same distribution for the logical volume hd6. The `lslv` command also shows that it has 288 LPs but no additional copies. It also says that the intra-policy of center is only 37 percent in band, which means that 63 percent is out of band, that is, not in the center. This we know from the previous `lspv` command to be true, since 108+72 LPs are placed on the inner and outer middle respectively and only 108 LPs are actually placed in the center; thus, 108/288 is 37.5 percent.

```
# print "scale=3\n(108/288)*100"|bc
37.500
```

lsvg

The `lsvg` command displays information about volume groups. When information from the Device Configuration database is unavailable, some of the fields will contain a question mark (?) in place of the missing data.

First, we need to understand the basic properties of the volume group, such as its general characteristics, its currently allocated size, its PP size, if there are any STALE partitions, and how much space is and is not occupied (in case we need to reorganize for performance reasons). For example:

```
# lsvg  -L vg99
VOLUME GROUP:   vg99                     VG IDENTIFIER:  006015611f031daa
VG STATE:       active                   PP SIZE:        64 megabyte(s)
VG PERMISSION:  read/write               TOTAL PPs:      543 (34752 megabytes)
MAX LVs:        256                      FREE PPs:       525 (33600 megabytes)
LVs:            2                        USED PPs:       18 (1152 megabytes)
OPEN LVs:       2                        QUORUM:         2
TOTAL PVs:      1                        VG DESCRIPTORS: 2
STALE PVs:      0                        STALE PPs:      0
ACTIVE PVs:     1                        AUTO ON:        yes
MAX PPs per PV: 1016                     MAX PVs:        32
```

The volume group has only two logical volumes and one disk with a PP size of 64 MB.

Second, we need to find out which logical volumes are created on this volume group and if they all are open and used; if they are not open and used, they might be old, corrupted and forgotten or only used occasionally, and if we were to need more space to reorganize the volume group, we might be able to free that space. For example:

```
# lsvg  -l vg99
vg99:
LV NAME             TYPE       LPs    PPs    PVs  LV STATE      MOUNT POINT
loglv00             jfslog     1      1      1    open/syncd    N/A
lv02                jfs        17     17     1    open/syncd    /testfs
```

As the preceding example shows, there is only one filesystem with a JFS log allocated on the entire volume group. Remember that the PP size was 64 MB, so even though the JFS log only has 1 LP/PP, it is a 64 MB partition.

We would also like to know which disks are allocated for this volume group. For example:

```
# lsvg  -p vg99
vg99:
PV_NAME           PV STATE    TOTAL PPs   FREE PPs    FREE DISTRIBUTION
hdisk6            active      543         525         109..91..108..108..109
```

There is only one disk in this volume group and no mirroring is being done for the logical volumes, but what kind of disk is it? Use the following command to discover this information:

```
# lsdev -Cl hdisk6
hdisk6 Available N0-58-L SSA Logical Disk Drive
```

It is an SSA logical disk let us find out more with the following command:

```
# ssaxlate -l hdisk6
pdisk0 pdisk2 pdisk1 pdisk3
```

The logical disk hdisk6 is composed of four physical disks (pdisk0-3) and there should be some sort of SSA RAID configuration (since the hdisks consists of more than one pdisk); to find out, we use the following command:

```
# ssaraid -M|xargs -i ssaraid -l {} -Ihz -n hdisk6
#name          id              state                   size

hdisk6         156139E312C44CO good                    36.4GB  RAID-10 array
```

It confirmed that it was a RAID definition.

To find out all SSA configured RAID disks controlled by SSA RAID managers, run the following command:

```
# ssaraid -M|xargs -i ssaraid -l {} -Ihz
#name          id              use         member_stat size

pdisk0         000629D148ED00D member      n/a         18.2GB  Physical disk
pdisk1         000629D2781600D member      n/a         18.2GB  Physical disk
pdisk2         000629D278C500D member      n/a         18.2GB  Physical disk
pdisk3         000629D282C500D member      n/a         18.2GB  Physical disk
hdisk6         156139E312C44CO good                    36.4GB  RAID-10 array
```

getlvcb

The `getlvcb` command is useful when you want to extract logical volume information and use this information in shell scripts. To understand what information can be extracted by `getlvcb`, use the -AT flags:

```
# getlvcb -AT lv99
```

The next example shows how `getlvcb` can be used in a shell script to extract the logical volume id and logical volume type:

```
#!/bin/ksh

lsvg|xargs -i lsvg -l {}|awk '$3~/[0-9]/{print $1}'|
while read lv;do
  printf "%-12.12s %-20.20s %s\n" $lv $(getlvcb -i -t $lv)
done|sort -k1
```

This is a sample output with `rootvg` and a second volume group that only contains two logical volumes (highlighted) if the above script was named `lslvcb` (compare this output to `lsvg|xargs -i getlvodm -L {}`). The output could look something like the following when the `lslvcb` script is run:

```
# lslvcb
hd1          00601561fb9811c0.8   jfs
hd2          00601561fb9811c0.5   jfs
hd3          00601561fb9811c0.7   jfs
hd4          00601561fb9811c0.4   jfs
hd5          00601561fb9811c0.1   boot
hd6          00601561fb9811c0.2   paging
hd7          00601561fb9811c0.10  sysdump
hd8          00601561fb9811c0.3   jfslog
hd9var       00601561fb9811c0.6   jfs
loglv00      006015611f031daa.1   jfslog
lv00         00601561fb9811c0.9   sysdump
lv01         00601561fb9811c0.11  jfs
lv02         006015611f031daa.2   jfs
```

11.1.15 mmlsmgr/mmdf/mmlsfs/mmlsdisk/mmfsadm

The `mm` commands operate on GPFS objects, and a few of the monitoring commands are exemplified here.

mmlsmgr

The `mmlsmgr` command (new with GPFS 1.3) displays which node is the GPFS File System Manager for a specific file systems. If no file system is specified, information about all file systems, for which a GPFS File System Manager is appointed, is displayed. For example:

```
# mmlsmgr
file system      manager node
---------------- ------------------
gpfsdev          5 (sp6n05)
```

mmdf

The `mmdf` command queries available file space on a GPFS file system. For example:

```
# mmdf gpfsdev
disk            disk size  failure holds    holds         free KB         free KB
name                in KB    group metadata data   in full blocks    in fragments
--------------- --------- -------- -------- ----- --------------- ---------------
gpfs0vsd          8880128     4005 yes      yes     8852480 (100%)      1216 ( 0%)
gpfs1vsd          8880128     4007 yes      yes     8854016 (100%)       848 ( 0%)
                ---------                          -------------- --------------
(total)          17760256                          17706496 (100%)      2064 ( 0%)

Inode Information
------------------
Total number of inodes: 32768
Total number of free inodes: 32113
```

mmlsfs

The `mmlsfs` command displays GPFS file system attributes. For example:

```
# mmlsfs  gpfsdev -d -s
flag value          description
---- -------------- -----------------------------------------------------
 -d  gpfs0vsd;gpfs1vsd  Disks in file system
 -s  roundRobin     Stripe method
```

mmlsdisk

The `mmlsdisk` command displays the current state of disks in a file system. For example:

```
# mmlsdisk  gpfsdev
disk         driver   sector failure holds    holds
name         type       size   group metadata data  status        availability
------------ -------- ------ ------- -------- ----- ------------- ------------
gpfs0vsd     disk        512    4005 yes      yes   ready         up
gpfs1vsd     disk        512    4007 yes      yes   ready         up
```

mmfsadm

The `mmfsadm` command is not a user supported command, and as is such subject to change or deletion without notice from IBM. However, we found it most useful for some low level monitoring of VSD performance and utilization. It should be executed on a VSD node. The `dump` option, along with `tscomm`, cfgmgr, stripe, disk, malloc and pgalloc, are some of the more useful suboptions. The following shows all configured nodes in a list:

```
# mmfsadm dump cfgmgr|grep -p '  node'
     node                 adapter     admin         fails   SGs  mem daem TMreq
 idx   no host name       ip address   node  status panics mngd free CPU   /sec
---- ---- -------------- ----------- ----- ------- ------- --- ---- ---- -----
   0    5 sp6n05         192.168.16.5    y      up   0/0     0   0% 100%     0
   1    7 sp6n07         192.168.16.7    y      up   0/0     0   0% 100%     0
   2    1 sp6n01         192.168.16.1    y      up   0/0     0   0% 100%     0
   3    3 sp6n03         192.168.16.3    y      up   0/0     0   0% 100%     0
   4    9 sp6n09         192.168.16.9    y      up   0/0     0   0% 100%     0
   5   10 sp6n10         192.168.16.10   y    down   0/0     0   0% 100%     0
   6   11 sp6n11         192.168.16.11   y      up   0/0     0   0% 100%     0
   7   12 sp6n12         192.168.16.12   y      up   0/0     0   0% 100%     0
   8   13 sp6n13         192.168.16.13   y      up   0/0     0   0% 100%     0
   9   14 sp6n14         192.168.16.14   y      up   0/0     0   0% 100%     0
```

11.1.16 ndb

The ndb command can be useful sometimes to examine the runtime status of sockets and interface layers. It can also be accessed through the crash command. The following example shows how to extract information about the first network interface:

```
# print "ndd\nquit\nquit"|ndb|sed '/^$/d' &
> netstat -I en3 &
> entstat ent3|grep "S/W Transmit Queue"
address of tcb is 0x0504a480
address of udb is 0x050a2680
type ? for help
ndb> address of ndd is 0x0024bf58
    ---------------------- NDD INFO -----(@0x50f3f020)----------
    name: ent3           alias: en3     ndd_next:0x5003d820
    flags:0x0002091b    (UP|BROADCAST|RUNNING|NOECHO|ALT ADDRS|64BIT|PSEG)
    ndd_open():  0x013eb2c4  ndd_close():0x013eb324  ndd_output():0x013eb330
    ndd_ctl():   0x013eb33c  ndd_stat(): 0x013ede68  receive():   0x013ede5c
    ndd_correlator: 0x50f3f000   ndd_refcnt:         1
    ndd_mtu:            1514     ndd_mintu:          60
    ndd_addrlen:           6     ndd_hdrlen:         14
<<< lines omitted >>>
    ndd_ipackets:        3249863    ndd_opackets:          4101666
    ndd_ierrors:               0    ndd_oerrors:                 0
    ndd_ibytes:        361081621    ndd_obytes:         2091025199
    ndd_recvintr:        3241114    ndd_xmitintr:            22410
    ndd_ipackets_drop:         0    ndd_nobufs:                  0
    ndd_xmitque_max:          79    ndd_xmitque_ovf:             0
ndb.ndd>
ndb>
Exiting.
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
en3   1500  link#3      0.4.ac.3e.c8.ca    3249792     0  4101668     0     0
en3   1500  129.40.35.6 night4cw.ppd.pok.  3249792     0  4101668     0     0
Max Packets on S/W Transmit Queue: 79
S/W Transmit Queue Overflow: 0
```

As can be seen in the preceding example output, by running `ndb`, `netstat` and `entstat`, important values are matching each other (and have been highlighted).

The following examples of `ndb` are based on the following output from `netstat`:

```
# netstat -A|grep -p "Active Internet"|head -3
Active Internet connections
PCB/ADDR Proto Recv-Q Send-Q  Local Address      Foreign Address    (state)
7042dedc tcp4       0      2  night4n37.login    night4cw.1023      ESTABLISHED
```

First, we use the `socket` subcommand with `ndb`. For example:

```
# print "socket 707ccadc\nquit"|ndb|sed '/^$/d'
address of tcb is 0x050f6980
address of udb is 0x0514eb80
type ? for help
ndb>     --------------------- SOCKET INFO -------------------
    type:0x7042 (BOGUS)    opts:0xffffdedc  (REUSEADDR|KEEPALIVE|DONTROUTE|USELO
OPBACK|LINGER|REUSEPORT|USE_IFBUFS|CKSUMRECV|NOREUSEADDR)
    state:0xffffdedc  (ISCONNECTING|ISDISCONNECTING|CANTSENDMORE|RCVATMARK|PRIV|
ASYNC|SS_ISCONFIRMING)
    linger:0x7042  pcb:0x00040000  proto:0x00030000
    q0:0x00000000  q0len:    0 q:0x000005b4
    qlen:    0  qlimit:    0  head:0x00000003
    timeo:    0  error:    0  oobmark:2166891387  pgid:2166891366
    snd: cc:2641404642  hiwat:2641404612  mbcnt:3020192430  mbmax:2641404612
    lowat:2639834263  mb:0x00003ebc  events:0xb404
    iodone:0xb40461d5  ioargs:0x00000000     flags:0x0000  ()
    rcv: cc:33620991  hiwat:    0  mbcnt:    0  mbmax:1342177280
    lowat:    0  mb:0x00000000  events:0x   0
    iodone:0x0000ffff  ioargs:0x81282366     flags:0xffff8128  (SEL|COLL|SB_WAIT
ING)
ndb>
Exiting.
```

The socket options that the application is using at the time the `ndb` command checked are highlighted in the preceding example. Then we use the `tcb` subcommand with `ndb`. For example:

```
# print "tcb 702dfedc\nquit"|ndb|sed '/^$/d'
ndb: error reading input, goodbye.
address of tcb is 0x0504a480
address of udb is 0x050a2680
type ? for help
ndb>     (TCB)---------------- INPCB  INFO -------------------
    next:0x702dfedc    prev:  0x702dfedc    head: 0x00040000
    ppcb:0x00000000    socket:0x1770dee9
    flags:0x09721216  (RECVRETOPTS|RECVDSTADDR|RECVIFINFO|HDRINCL)
    iflowinfo:0x00000000 (UNKNOWN)
    oflowinfo:0x00000000 (UNKNOWN)
    tos:  0 (UNKOWN)
    ttl:  0
    lport: 2418   laddr:0x09721216 (nighthawk4.ppd.pok.ibm.com)
    fport:    0   faddr:0x000005ac (NONE)
    laddr6:(::6:14:972:1216)
    faddr6:(2dd7:0:0:3::5ac)
    --------------------- SOCKET INFO -------------------
<<< lines omitted >>>
```

This next example shows how to use `pstat` to dump the system file table:

```
# pstat -f
FILE TABLE:

 SLOT    FILE      REF     TYPE            ADDRESS      FLAGS
   0  0x10000000     4  inode.VREG        0x15169a10   0x00001001 read rshare
   1  0x10000030     1  inode.VREG        0x140b0e90   0x00000020 exec
   2  0x10000060    55  inode.VREG        0x1342d130   0x00001001 read rshare
   3  0x10000090     1  socket.DGRAM      0x7017f800   0x00000003 read write
   4  0x100000c0     1  inode.VCHR        0x526ff070   0x00000002 write
   5  0x100000f0    51  inode.VREG        0x1836ff30   0x00001001 read rshare
<<< lines omitted >>>
  25  0x100004b0     3  socket.STREAM     0x707b1000   0x00000007 read write nonblock
<<< lines omitted >>>
  33  0x10000630     1  socket.DGRAM      0x7089fa00   0x00028003 read write ndelay as
<<< lines omitted >>>
```

We select two sockets from the output above (number 25 and number 33) and we will show how to find out more about them with ndb. The first example is the TCP (STREAM) socket (number 25):

```
# print "socket 707b1000\nquit"|ndb|sed '/^$/d'
address of tcb is 0x050f6980
address of udb is 0x0514eb80
type ? for help
ndb>     --------------------- SOCKET INFO -------------------
    type:0x0001 (STREAM)    opts:0x010c  (REUSEADDR|KEEPALIVE|OOBINLINE)
    state:0x0102  (ISCONNECTED|NBIO)
    linger:0x0000  pcb:0x707b1244  proto:0x050f4220
    q0:0x00000000  q0len:    0 q:0x00000000
    qlen:    0  qlimit:    0  head:0x00000000
    timeo:    0  error:    0  oobmark:    0  pgid:    0
    snd: cc:    2  hiwat:64240  mbcnt:  276  mbmax:256960
    lowat: 4096  mb:0x70411e00  events:0x   0
    iodone:0x00000000  ioargs:0x00000000     flags:0x0048  (SEL|NOINTR)
    rcv: cc:    0  hiwat:64240  mbcnt:    0  mbmax:256960
    lowat:    1  mb:0x00000000  events:0x   0
    iodone:0x00000000  ioargs:0x00000000     flags:0x0048  (SEL|NOINTR)
ndb>
```

What is interesting to watch is the option settings for open sockets; in some cases, applications are not affected by system changes with the no command, but uses setsockopt() to override the default socket settings.

The meaning of the above socket options are:

KEEPALIVE Monitors the activity of a connection by enabling or disabling the periodic transmission of ACK messages on a connected socket. The idle interval time can be designated using the no command.

OOBINLINE Leaves received out-of-band data (data marked urgent) in line.

REUSEADDR Specifies that the rules used in validating addresses supplied by a bind subroutine should allow reuse of a local port. SO_REUSEADDR allows an application to explicitly deny subsequent bind subroutine to the port/address of the socket with SO_REUSEADDR set. This allows an application to block other applications from binding with the bind() subroutine.

Remember that the TCP protocol level socket options are inherited from listening sockets to new sockets. The following TCP options can be changed by the application for an open socket; in case of unexpected communications performance for the application, it is important to know about the settings:

TCP_RFC1323 Enables or disables RFC 1323 enhancements on the specified TCP socket.

TCP_NODELAY Specifies whether TCP should follow the Nagle Algorithm for deciding when to send data. By default, TCP will follow the Nagle Algorithm. To disable this behavior, applications can enable TCP_NODELAY to force TCP to always send data immediately.

TCP_STDURG Enables or disables RFC 1122 compliant urgent point handling. By default, TCP implements urgent pointer behavior compliant with the 4.2 BSD operating system; this option defaults to 0.

This next example is the output from `ndb` for the selected UDP (DGRAM) socket (number 33):

```
# print "socket 7089fa00\nquit"|ndb|sed '/^$/d'
address of tcb is 0x050f6980
address of udb is 0x0514eb80
type ? for help
ndb>     --------------------- SOCKET INFO -------------------
    type:0x0002 (DGRAM)    opts:0x0020  (BROADCAST)
    state:0x0380  (PRIV|NBIO|ASYNC)
    linger:0x0000  pcb:0x70877f00  proto:0x050f4788
    q0:0x00000000  q0len:    0 q:0x00000000
    qlen:    0  qlimit:    0  head:0x00000000
    timeo:    0  error:    0  oobmark:    0  pgid:4294960668
    snd: cc:    0  hiwat:32768  mbcnt:    0  mbmax:131072
    lowat: 4096  mb:0x00000000  events:0x   0
    iodone:0x00000000  ioargs:0x00000000     flags:0x0010  (ASYNC)
    rcv: cc:    0  hiwat:65536  mbcnt:    0  mbmax:262144
    lowat:    1  mb:0x00000000  events:0x   0
    iodone:0x00000000  ioargs:0x00000000     flags:0x0010  (ASYNC)
ndb>
Exiting.
```

The meaning of the above socket option is:

BROADCAST Permits sending of broadcast messages.

11.1.17 netpmon

`netpmon` monitors a trace of system events and reports on CPU usage, network device-driver I/O, internet socket calls, and NFS I/O. In its normal mode, `netpmon` runs in the background while one or more application programs or system commands are being executed and monitored. When tracing is stopped via a `trcstop` command, `netpmon` generates all specified reports and exits.

Note

`netpmon` provides only partial support on most (PowerPC systems) PCI-based network adapters, and `netpmon` does *not* support NFS Version 3 metrics (NFS3/ONC+).

The following example shows how to run `netpmon`. To stop `netpmon` tracing of I/O, `trcstop` must be issued, and it is when this is done that `netpmon` writes the output. To have `netpmon` monitor network I/O during a time interval, just run the `sleep` program with the specified amount of seconds and then the `trcstop` program. For example:

```
# netpmon -o netpmon.out -O all -tv && sleep 60;trcstop

Enter the "trcstop" command to complete filemon processing

[netpmon: Reporting started]

[netpmon: Reporting completed]

[netpmon: 60.583 secs in measured interval]
```

The output from `netpmon` can be quite extensive; to quickly find out if something is in need of attention, we filter it with the `awk` command in most of the following examples below in order to extract specific summary tables from the `netpmon` output file. We have used the `all` option, but we could have used any of these options:

cpu — CPU usage

dd — Network device-driver I/O

so — Internet socket call I/O

nfs — NFS I/O

all — Short for cpu, dd, so, nfs

Additional parameters that could be interesting to apply are:

-t — Prints CPU reports on a per-thread basis.

-v Prints extra information in the report. All processes and all accessed remote files are included in the report, instead of only the 20 most active processes and files.

When analyzing the output, keep in mind that the avg (average) measurement is an indication of efficiency and that the sdev measurement indicates the level of fragmentation (standard deviation).

Network device-driver transmit statistics/device

In the following example, we only extract the device driver transmit statistics divided per adapter interface:

```
# awk '/^Network.*by Device/,/^\=/' netpmon.out|sed '/^$/d;/^=/d'
Network Device-Driver Statistics (by Device):
-------------------------------------------
                       ----------- Xmit -----------   -------- Recv ---------
Device                   Pkts/s  Bytes/s  Util  QLen   Pkts/s  Bytes/s   Demux
------------------------------------------------------------------------------
token ring 0              74.32     8939  0.0%39.136    77.99     7989  0.0298
ethernet 0               285.06    27970  0.8% 0.008   285.11    27977  0.0309
```

To interpret the above output, use the following descriptions:

Xmit Pkts/s	Packets per second transmitted through this device.
Xmit Bytes/s	Bytes per second transmitted through this device.
Xmit Util	Busy time for this device, as a percent of total time.
Xmit Qlen	Number of requests waiting to be transmitted through this device, averaged over time, including any transaction currently being transmitted.
Recv Pkts/s	Packets per second received through this device.
Recv Bytes/s	Bytes per second received through this device.
Recv Demux	Time spent in demux layer as a fraction of total time.

Network device-driver transmit statistics/destination

In the next example, we only extract the device driver transmit statistics divided per destination:

```
# awk '/^Network.*by Destination/,/^\=/' netpmon.out|sed '/^$/d;/^=/d'
Network Device-Driver Transmit Statistics (by Destination Host):
----------------------------------------------------------------
Host                    Pkts/s  Bytes/s
----------------------------------------
sp6n07                  275.69    27018
9.12.0.119                0.28      416
9.12.0.114                1.54      334
sp6n14                    1.40      165
```

To interpret the above output, use the following descriptions:

Host	Destination host name. An * (asterisk) is used for transmissions for which no host name can be determined.
Pkts/s	Packets per second transmitted to this host.
Bytes/s	Bytes per second transmitted to this host.

Detailed protocol

In the following example, we only extract the detailed protocol information:

```
# grep -p ^PROTOCOL netpmon.out
PROTOCOL: TCP (All Processes)
reads:                  3598
  read sizes (bytes):   avg 4090.3  min 1       max 4096    sdev 152.5
  read times (msec):    avg 0.180   min 0.003   max 157.295 sdev 4.340
writes:                 20
  write sizes (bytes):  avg 40.6    min 5       max 145     sdev 60.3
  write times (msec):   avg 0.018   min 0.008   max 0.050   sdev 0.014
```

To interpret the above output, use the following descriptions:

Reads	Number of read(), recv(), recvfrom(), and recvmsg() subroutines made by this process on sockets of this type.
Read Sizes (Bytes)	Size statistics for read() calls.
Read Times (Msec)	Response time statistics for read() calls.
Writes	Number of write(), send(), sendto(), and sendmsg() subroutines made by this process on sockets of this type.
Write Sizes (Bytes)	Size statistics for write() calls.

`Write Times (Msec)` Response time statistics for `write()` calls.

Detailed interface

In the following example, we only extract the detailed interface information:

```
# grep -p ^DEVICE netpmon.out
DEVICE: token ring 0
recv packets:          38
  recv sizes (bytes):  avg 62.3    min 62     max 64      sdev 0.5
  recv times (msec):   avg 0.034   min 0.027  max 0.042   sdev 0.005
  demux times (msec):  avg 3.145   min 0.129  max 6.332   sdev 2.729
xmit packets:          13
  xmit sizes (bytes):  avg 412.2   min 64     max 1502    sdev 463.8
  xmit times (msec):   avg 2626.442 min 0.535  max 4151.737 sdev 1807.529

DEVICE: ethernet 0
recv packets:          1978
  recv sizes (bytes):  avg 98.1    min 98     max 118     sdev 1.4
  recv times (msec):   avg 0.023   min 0.019  max 0.138   sdev 0.003
  demux times (msec):  avg 0.106   min 0.070  max 0.148   sdev 0.004
xmit packets:          1978
  xmit sizes (bytes):  avg 98.1    min 98     max 118     sdev 1.4
  xmit times (msec):   avg 0.027   min 0.025  max 0.057   sdev 0.001
```

To interpret the above output, use the following descriptions:

Recv Packets — Number of packets received through this device.

Recv Sizes (Bytes) — Size statistics for received packets.

Recv Times (msec) — Response time statistics for processing received packets.

Xmit Packets — Number of packets transmitted to this host.

Demux Times (msec) — Time statistics for processing received packets in the demux layer.

Xmit Sizes (Bytes) — Size statistics for transmitted packets.

Xmit Times (Msec) — Response time statistics for processing transmitted packets.

Detailed host

In the following example, we only extract the detailed host information:

```
# grep -p ^HOST netpmon.out
HOST: sp6n07
xmit packets:          7251
  xmit sizes (bytes):   avg 98.0    min 98      max 98      sdev 0.0
  xmit times (msec):    avg 0.027   min 0.025   max 0.155   sdev 0.003
<<< lines omitted >>>
```

To interpret the above output, use the following descriptions:

Host	Destination host name.
Xmit Packets	Number of packets transmitted through this device.
Xmit Sizes (Bytes)	Size statistics for transmitted packets.
Xmit Times (Msec)	Response time statistics for processing transmitted packets.

Summary table for socket call statistics

In the following example, we only extract the summary table for socket calls:

```
# grep -p '^TCP Socket Call' netpmon.out
TCP Socket Call Statistics (by Process):
----------------------------------------
                                   ------ Read -----   ----- Write -----
Process                       PID  Calls/s   Bytes/s   Calls/s   Bytes/s
------------------------------------------------------------------------
rsh                         20142    26.58    108770      0.10         4
    Thread id:              28671    26.58    108770      0.10         4
rsh                         20650    26.27    107500      0.10         4
    Thread id:              29413    26.27    107500      0.10         4
rsh                         17768    25.91    106019      0.10         4
    Thread id:              25569    25.91    106019      0.10         4
rsh                         19912    14.16     57877      0.10         4
    Thread id:              28911    14.16     57877      0.10         4
rsh                         19552     0.03         0      0.10         4
    Thread id:              29177     0.03         0      0.10         4
------------------------------------------------------------------------
Total (all processes)                92.94    380167      0.52        21
```

To interpret the above output use the following descriptions, (note that the above output is on a thread basis due to our usage of the `-t` option):

Read Calls/s	Number of read(), recv(), and recvfrom() subroutines per second made by this process on sockets of this type.
Read Bytes/s	Bytes per second requested by the above calls.
Write Calls/s	Number of write(), send(), and sendto() subroutines per second made by this process on sockets of this type.

Write Bytes/s   Bytes per second written by this process to sockets of this protocol type.

11.1.18 netstat

Traditionally, `netstat` is used for determining network problems rather than for measuring performance. But it is very useful in determining the amount of traffic on the network and therefore ascertain whether performance problems are due to congestion or limitations in allocations of logical resources. For network performance analysis, the `netstat` tool is for network tuning what `vmstat` is for basic system tuning.

Among the many types of information that `netstat` will provide, the following are some of the most important for network performance analysis:

- Communication subsystem summaries
- Active connections
- Routing table
- Adapter and network statistics
- Network memory usage
- Protocol statistics

Note

Before using `netstat` to continually monitor network traffic and network resource usage, reset the counters with the following command:

```
# netstat -Zc -Zi -Zm -Zs
```

Communication subsystem summaries

The following example shows the number of packets received, transmitted, and dropped in the communications subsystem. In the statistics output, a N/A displayed in a field value indicates the count is not applicable. For the NFS/RPC statistics, the number of incoming packets that pass through RPC are the same packets which pass through NFS, so these numbers are not summed in the NFS/RPC Total field, thus the N/A. NFS has no outgoing packet or outgoing packet drop counters specific to NFS and RPC. Therefore, individual counts have a field value of N/A, and the cumulative count is stored in the NFS/RPC Total field.

```
# netstat -D

Source                         Ipkts                Opkts     Idrops     Odrops
-------------------------------------------------------------------------------
ent_dev0                     3929421              1223715          0          0
               ----------------------------------------------------------------
Devices Total                3929421              1223715          0          0
-------------------------------------------------------------------------------
ent_dd0                      3929421              1223716          0          1
               ----------------------------------------------------------------
Drivers Total                3929421              1223716          0          1
-------------------------------------------------------------------------------
ent_dmx0                     3929421                  N/A          0        N/A
               ----------------------------------------------------------------
Demuxer Total                3929421                  N/A          0        N/A
-------------------------------------------------------------------------------
IP                           4083011              4081590       1385          0
TCP                          3698997              1017417          0          0
UDP                           310112               288510        151          0
               ----------------------------------------------------------------
Protocols Total              8092120              5387517       1536          0
-------------------------------------------------------------------------------
lo_if0                         18468                18475          7          0
en_if0                       3929421              1223712          0          0
css_if0                       136438               153834          0          0
               ----------------------------------------------------------------
Net IF Total                 4084327              1396021          7          0
-------------------------------------------------------------------------------
               ----------------------------------------------------------------
NFS/RPC Total                    N/A               391142          0          0
-------------------------------------------------------------------------------
(Note:  N/A -> Not Applicable)
```

Active connections

The following example shows all active internet connections (including servers) that are ESTABLISHED:

```
# netstat -a|awk '/^$/{exit}{print}'|grep ESTABLISHED
tcp4       0      0  sp6n07.ftp       sp6en0.42593    ESTABLISHED
tcp4       0      0  sp6sw07.36349    sp6sw01.6667    ESTABLISHED
tcp4       0      0  sp6n07.login     sp6en0.1023     ESTABLISHED
tcp4       0      0  sp6n07.telnet    sp6en0.41325    ESTABLISHED
tcp4       0      0  loopback.smux    loopback.32845  ESTABLISHED
tcp4       0      0  loopback.32845   loopback.smux   ESTABLISHED
tcp4       0      0  loopback.smux    loopback.32769  ESTABLISHED
tcp4       0      0  loopback.32769   loopback.smux   ESTABLISHED
```

The following example also shows active internet connections:

```
# netstat -A|grep -p "Active Internet"
Active Internet connections
PCB/ADDR Proto Recv-Q Send-Q  Local Address      Foreign Address    (state)
7042dedc tcp4       0      2  night4n37.login    night4cw.1023      ESTABLISHED
707ccadc tcp4       0      0  loopback.smux      loopback.32849     ESTABLISHED
707cc6dc tcp4       0      0  loopback.32849     loopback.smux      ESTABLISHED
7082a6dc tcp4       0      0  loopback.smux      loopback.32770     ESTABLISHED
7082aadc tcp4       0      0  loopback.32770     loopback.smux      ESTABLISHED
70877c00 udp4       0      0  129.40.35.12.ntp   *.*
70877d00 udp4       0      0  127.255.255..ntp   *.*
70877f00 udp4       0      0  night4n37.ntp      *.*
7017db00 udp4       0      0  loopback.ntp       *.*
```

This next example shows active internet connections that are not ESTABLISHED. If they are in the LISTEN state, it means that it's a server process that is waiting for a client process to open communication with it (for many sockets, it is the inetd daemon that is listening). Unspecified addresses and ports appear as an * (asterisk) in the output.

```
# netstat -a|awk '$6!=""/^$/{exit}{print}'|grep -v ESTABLISHED
Proto Recv-Q Send-Q  Local Address Foreign Address (state)
tcp4       0      0  *.6668        *.*             LISTEN
tcp4       0      0  *.6667        *.*             LISTEN
tcp4       0      0  *.sysctl      *.*             LISTEN
tcp4       0      0  *.writesrv    *.*             LISTEN
tcp4       0      0  *.spseccfg    *.*             LISTEN
tcp4       0      0  *.time        *.*             LISTEN
tcp4       0      0  *.daytime     *.*             LISTEN
tcp4       0      0  *.chargen     *.*             LISTEN
tcp4       0      0  *.discard     *.*             LISTEN
tcp4       0      0  *.echo        *.*             LISTEN
tcp        0      0  *.exec        *.*             LISTEN
tcp4       0      0  *.klogin      *.*             LISTEN
tcp        0      0  *.login       *.*             LISTEN
tcp4       0      0  *.kshell      *.*             LISTEN
tcp        0      0  *.shell       *.*             LISTEN
tcp        0      0  *.telnet      *.*             LISTEN
tcp        0      0  *.ftp         *.*             LISTEN
tcp4       0      0  *.32771       *.*             LISTEN
tcp4       0      0  *.652         *.*             LISTEN
tcp4       0      0  *.651         *.*             LISTEN
tcp4       0      0  *.32770       *.*             LISTEN
tcp4       0      0  *.smux        *.*             LISTEN
tcp4       0      0  *.sunrpc      *.*             LISTEN
tcp4       0      0  *.smtp        *.*             LISTEN
```

Routing table

Since performance problems can occur with NFS servers and name resolution and routing, it can be important to check the current routing table with `netstat`. When *dynamic mtu discovery* is enabled, the routing table can fill up very fast on a busy server; in this case, do not use name resolution when running `netstat`, as in the following example (however, this example is normal):

```
# netstat -rnf inet
Routing tables
Destination      Gateway           Flags   Refs     Use  If   PMTU  Exp  Groups

Route Tree for Protocol Family 2 (Internet):
default          129.40.35.123     UGc       0        0  en0     -   -
9.114.18.22      129.40.35.123     UGHW      3    10882  en0  1500   -
127/8            127.0.0.1         U         7      911  lo0     -   -
129.10.10/24     129.10.10.37      U         1   157483 css0     -   -
129.40.35.64/26  129.40.35.102     U        15  1225073  en0     -   -
```

Adapter and network statistics

To display information about the adapter queues and other network specific statistics, use the `netstat -v` command. This invokes the appropriate stat command (`entstat`/`tokstat`/`atmstat`/`fddistat`) for each adapter type. In the following output example, we have extracted some of the information that is important to look at:

```
<<< lines omitted >>>
Transmit Errors: 0                          Receive Errors: 0
Packets Dropped: 1                          Packets Dropped: 0
                                            Bad Packets: 0
Max Packets on S/W Transmit Queue: 62
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 1
<<< lines omitted >>>
```

Obviously Receive/Transmit Errors are important, as are Bad/Dropped Packets. The adapter device drivers buffer queue (list pointing to mbufs) will show as Max Packets, which is the maximum number of packets that has been on the queue at the same time. Queue Overflow is, of course, how many times there have been requests to transmit packets that could not be performed because the adapters queue was already full. And finally we have the Current field, which is self explanatory.

It is also interesting to look at transfer collisions for Ethernet. At the end of the listing, there is a Packets with Transmit collisions table. The first position is the same as the field Single Collision Count and the rest are 2 to 16 number of times the collision detection *backoff* has been performed for sending packets (a summary is in the Multiple Collision Count field). Since the time increases for each additional packet transmission, network traffic is degraded if multiple collisions occur for the same packet and are occurring frequently.

In some environments, Ethernet adapters driven above 25 percent to 30 percent will result in serious throughput degradation (it usually starts at a 10 percent load). One indicator that the network is overloaded can be calculated by dividing the number of collisions with the number of transmitted packets, and if this is larger than 0.1, the network might be overloaded. In our next example, we use the following as input to our calculation:

```
Transmit Statistics:                        Receive Statistics:
--------------------                        -------------------
Packets: 1852749                            Packets: 2319984
<<< lines omitted >>>
Single Collision Count: 83847
Multiple Collision Count: 92151
<<< lines omitted >>>
```

By applying our “rule of thumb” to the above values, as follows:

```
(83847+92151) / (1852749+2319984) = 0.042
```

We find that our network is not overloaded if we use this heuristic rule, because 0.042 is less than 0.1.

For Token-Ring, it can be interesting to note if the ring is actually running half or full duplex when full duplex is requested. This is shown by looking at the Media Speed Running field (compare with Media Speed Selected). The following example illustrates the dynamics of Token-Ring:

```
<<< lines omitted >>>
IBM PCI Tokenring Adapter (14103e00) Specific Statistics:
---------------------------------------------------------
Media Speed Running: 16 Mbps Half Duplex
Media Speed Selected: 16 Mbps Full Duplex
<<< lines omitted >>>
```

Network memory usage

To examine how the mbuf memory management is performing issue the `netstat -m` command. The following example extracts only the top part that contains a summary; in this example, we have *112 mbuf allocation failures*:

```
# netstat -m|head -5
4139 mbufs in use:
1034 Kbytes allocated to mbufs
112 requests for mbufs denied
0 calls to protocol drain routines
0 sockets not created because sockthresh was reached
```

To have the information displayed, as in the preceding example, you need to set the extendednetstats option to 1 with the `no` command in AIX 4.3.3[1].

```
# no -o extendednetstats=1
```

[1] The change is effective immediately.

The following example extracts the relevant part for CPU 0:

```
# netstat -m|awk 'BEGIN{go=0}go>0&&/^$/{exit}go>1{print}/CPU 0/,/By/{print;go++}'
******* CPU 0 *******
By size       inuse     calls failed   delayed    free   hiwat   freed
32              137       305      0         0     119    2320       0
64               48       174      0         0      16    1160       0
128              34      2788      0         0      62     580       0
256            5996     41812     15         0    1364    1392       0
512              47       487      0         0       9     145       0
1024              7       131      0         0       5     362       0
2048           2048      2063    112         0       2     362       0
4096              3         8      0         0       2     435       0
8192              0         6      0         0       0      36       0
16384             1       279      0         0      65      87       0
65536             1         1      0         0       0    4096       0
```

Put the following output in a script and call it status.mbuf.cpu.2 (before you run the script, make sure it is in your path) as follows; this script will use the first parameter as the CPU# and the second parameter as the delay factor in seconds:

```
#!/bin/ksh
no=${1:-0}
runrun()
{
  netstat -m|
  awk 'BEGIN{go=0}go>0&&/^$/{exit}go>1{print}/CPU '$no'/,/By/{print;go++}'
}
case $2 in
  "") runrun;;
  *) while :;do runrun;sleep $2;done;;
esac
```

This output can then be used in another script, as the following example shows, to examine each processors usage in a separate X-Windows `aixterm`:

```
#!/bin/ksh

integer nproc=$(LANG=C lsdev -Cc processor -S Available -Fname|wc -l)
integer i=0

[[ -f ${0##*/}.2 && ! -z "$DISPLAY" ]] || exit -1

while (($i<=$nproc-1));do
  aixterm -T "CPU $i" -e status.mbuf.cpu.2 $i 5 &
  ((i=$i+1))
done
```

Protocol statistics

`netstat` can also be used to monitor protocol specific statistics such as IP, ICMP, TCP and UDP. Use the -ss option and zero statistics will not be included. The following example illustrates the usage to monitor ICMP since we are concerned with erroneous redirect packets from some old OS/2 machines and want to find out if there are an excessive amount hitting our server node:

```
# netstat -ssp icmp
icmp:
        103 calls to icmp_error
        Output histogram:
                echo reply: 242
                destination unreachable: 103
                routing redirect: 97
        Input histogram:
                echo reply: 546
                destination unreachable: 715
                echo: 433
                time exceeded: 12
        242 message responses generated
```

In this case, we do not think that it was the case, since the number was achieved over more than a week and other systems that had been affected had much higher numbers for an equivalent time period.

The following example shows that UDP has dropped packets due to no socket:

```
# netstat -ssp udp
udp:
        623603 datagrams received
        37484 dropped due to no socket
        44 broadcast/multicast datagrams dropped due to no socket
        586075 delivered
        640024 datagrams output
```

If we have a server with a lot of connections, we want to monitor the IP queue, which we can do, as shown in this example:

```
# netstat -sp ip|grep ipintrq
         0 ipintrq overflows
```

If the `ipqintrq` shows a non zero value, IP connections can not be made to our node because the queue is to small for the demand. Check the current setting as follows with the `no` command:

```
# no -o ipqmaxlen
ipqmaxlen = 100
```

11.1.19 nfsstat

NFS gathers statistics on the types of NFS operations performed along with error information and performance indicators. You can use the `nfsstat` command to identify network problems and observe the amount of NFS operations taking place on the system.

Note

Before using `nfsstat` to continually monitor the NFS network traffic, reset the counters with the following command:

```
# nfsstat -z
```

When monitoring NFS statistics, be aware of how many biod threads each client has and how many nfsd threads the servers have (both the maximum and the currently active number of threads).

11.1.19.1 NFS Server

Examine the output from the `nfsstat` command on both the client and server nodes to find indications that there is too much NFS traffic generated for the server. The NFS server displays the number of NFS calls received (calls) and rejected (badcalls) due to authentication, as well as the counts and percentages for the various kinds of calls made.

To check how many nfsd threads are running in the current system, use the `ps` command as follows:

```
# ps -mo THREAD -p $(lssrc -s nfsd|awk '$1=="nfsd"&&/active/{print $3}')
    USER   PID  PPID    TID ST  CP PRI SC    WCHAN        F     TT BND COMMAND
    root  4478  9032      - A    0  60  3        *   240001      -   - /usr/sbin
/nfsd
       -     -     -   8529 S    0  60  1 50044608     1400      -   - -
       -     -     -  10857 S    0  60  1 5278cd98      400      -   - -
       -     -     -  11095 S    0  60  1   36c8bc     1400      -   - -
```

As can be seen in the preceding, there are only three threads active in this nfsd process, and as will be shown in the following, it is allowed to use up to 32 threads in the current sample setting.

To check the setting for the number of maximum threads, either check the nfsd process with the `ps` command and look at its arguments (it is usually started with the number of maximum threads as the parameter), as is shown in the following example:

```
# ps -e -F "args" |awk '/^\/.*nfsd/{print $2}'
32
```

You can also check the ODM, as is shown here:

```
# odmget -q subsysname=nfsd SRCsubsys|awk '/cmdargs/{print $3}'
"32"
```

The command line arguments overrides other settings when starting nfsd, but the nfso command has another limit that nfsd could use, the nfs_max_threads, and how to check it is shown here:

```
# nfso -o nfs_max_threads
nfs_max_threads= 8
```

The following example shows the output from the server part of the `nfsstat` command (-s):

```
# nfsstat -s
Server rpc:
Connection oriented
calls      badcalls   nullrecv   badlen     xdrcall    dupchecks  dupreqs
33650      0          0          0          0          8          0
Connectionless
calls      badcalls   nullrecv   badlen     xdrcall    dupchecks  dupreqs
6          0          0          0          0          0          0

Server nfs:
calls      badcalls   public_v2  public_v3
33656      0          0          0
Version 2: (0 calls)
null       getattr    setattr    root       lookup     readlink   read
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
wrcache    write      create     remove     rename     link       symlink
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
mkdir      rmdir      readdir    statfs
0 0%       0 0%       0 0%       0 0%
Version 3: (33656 calls)
null       getattr    setattr    lookup     access     readlink   read
114 0%     1012 3%    0 0%       509 1%     563 1%     0 0%       31352 93%
write      create     mkdir      symlink    mknod      remove     rmdir
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
rename     link       readdir    readdir+   fsstat     fsinfo     pathconf
0 0%       0 0%       0 0%       8 0%       5 0%       93 0%      0 0%
commit
0 0%
```

The preceding output displays a count of the various kinds of calls and their respective percentages. Some explanations of the output as shown above:

calls — The total number of RPC calls received from clients.

badcalls — The total number of calls rejected by the RPC layer.

nullrecv — The number of times an RPC call was not available when it was thought to be received. This used to be an indicator that there were too many nfsd processes started. However, because additional nfsd threads will now be started as they are needed, this should not be a concern.

badlen — Packets truncated or damaged; the number of RPC calls with a length shorter than a minimum-sized RPC call.

dupchecks The number of RPC calls that did a look-up in the duplicate request cache.

dupreqs The number of duplicate RPC calls found.

For example, if the percentage of getattr() calls is very high, then tuning attribute caches may be advantageous. If the percentage of write() calls is very high, then disk and LVM tuning is important. If the percentage of read() calls is very high, then using more RAM for caching files could improve performance, as could increasing maxperm with `vmtune` (see Section 6.1.2.6, “minperm and maxperm” on page 123).

11.1.19.2 NFS Client

The NFS client displays the number of calls sent and rejected, as well as the number of times a client handle was received (nclget), the number of times a call had to sleep while awaiting a handle (nclsleep), and a count of the various kinds of calls and their respective percentages.

To check how many biod threads are running in the current system, use the `ps` command as follows (remember that there is one additional kernel biod that is always running):

```
# ps -mo THREAD -p $(lssrc -s biod|awk '$1=="biod"&&/active/{print $3}')
    USER   PID  PPID    TID ST  CP PRI SC    WCHAN        F     TT BND COMMAND
    root  7486  4648      - A    0  60  1        -   240001      -   0 /usr/sbin
/biod
       -     -     -  18835 S    0  60  1        -      400      -   0 -
```

As can be seen in the preceding example, there is only one thread active in this biod process, and as will be shown in the following, it is allowed to use up to six threads in the current sample setting.

To check the setting for the number of maximum threads, check the biod process with the `ps` command, and look at its arguments (it is usually started with the number of maximum threads as the parameter), as shown in the following example:

```
# ps -e -F "args" |awk '/^\/.*biod/{print $2}'
6
```

You could also check the ODM, shown here:

```
# odmget -q subsysname=biod SRCsubsys|awk '/cmdargs/{print $3}'
"6"
```

NFS mountpoint

To query a mount point and its options on a client, issue the following `nfsstat` command (see also the `mount` command):

```
# nfsstat -m
/mnt from /mnt:night4n37
 Flags:   vers=3,proto=tcp,auth=unix,hard,intr,link,symlink,rsize=32768,wsize=32 768,re
 All:     srtt=0 (0ms), dev=0 (0ms), cur=0 (0ms)
```

Look at the line starting with All; the three groups give some quick performance related information, as follows:

srtt Smoothed round-trip time

dev Estimated deviation

cur Current backed-off time-out value

For an NFS client, if you see incidents of time-outs and retransmits, and the numbers are roughly equivalent, then you can be assured that there are packets being dropped. It will be of particular interest to monitor the RPC statistical data; the heading “Connection oriented“ refers to connections made with the TCP protocol and the heading “Connectionless“ refers to connections made with the UDP protocol:

```
# nfsstat -c
Client rpc:
Connection oriented
calls      badcalls   badxids    timeouts   newcreds   badverfs   timers
0          0          0          0          0          0          0
nomem      cantconn   interrupts
0          0          0
Connectionless
calls      badcalls   retrans    badxids    timeouts   newcreds   badverfs
20         0          0          0          0          0          0
timers     nomem      cantsend
0          0          0

Client nfs:
calls      badcalls   clgets     cltoomany
0          0          0          0
Version 2: (0 calls)
<<< lines omitted >>>
Version 3: (0 calls)
null       getattr    setattr    lookup     access     readlink   read
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
write      create     mkdir      symlink    mknod      remove     rmdir
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
rename     link       readdir    readdir+   fsstat     fsinfo     pathconf
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
commit
0 0%
```

The preceding output displays a count of the various kinds of calls and their respective percentages. Some explanations of the output shown above are:

calls	The total number of RPC calls made to NFS.
badcalls	The total number of calls rejected by the RPC layer.
retrans	The number of times a call had to be retransmitted due to a time-out while waiting for a reply from the server. This is applicable only to RPC over connection-less transports.
badxid	The number of times a reply from a server was received that did not correspond to any outstanding call. This means the server is taking too long to reply.
timeout	The number of times a call timed out while waiting for a reply from the server.
newcred	The number of times authentication information had to be refreshed.
nomem	The number of times a call failed due to a failure to allocate memory.
cantconn	The number of times a call failed due to a failure to make a connection to the server.

A network may drop a packet if it can not handle it. Dropped packets may be the result of the response time of the network hardware or software, or an overloaded CPU on the server. The severity of the performance degradation resulting from dropped packets is dependent on the number of packets dropped and the length of time during which the packets were dropped. The data from dropped packets is not actually lost because a replacement request is issued for them; they will, however, reduce performance because they will have to be retransmitted, thus causing a delay in delivery of the information they contained. See Chapter 4, “Network tuning” on page 21 for more information.

The retrans column in the RPC section displays the number of times requests were retransmitted due to a time-out in waiting for a response. This is related to dropped packets.

A high badxid count implies that requests are reaching the various NFS servers, but the servers are too busy to send replies before the client's RPC calls time out and are retransmitted. The badxid value is incremented each time a duplicate reply is received for a transmitted request (an RPC request retains its XID through all transmission cycles). Excessive retransmissions place an additional strain on the server, further degrading response time.

11.1.20 nslookup/host

The `nslookup` command queries domain name servers in two modes:

- interactive
- non-interactive

Interactive mode allows you to query name servers for information about various hosts and domains or to print a list of the hosts in a domain. In non-interactive mode, the names and requested information are printed for a specified host or domain.

The `host` command resolves a host name into an Internet address or an Internet address into a host name. If the local host is using DNS, the local or remote name server database is queried before searching the local /etc/hosts file.

The process of obtaining an Internet address from a host name is known as *name resolution*. The process of translating an Internet address into a host name is known as *reverse name resolution*. The presence of the /etc/resolv.conf file indicates that DNS should be used to resolve a name.

Resolver routines on hosts running TCP/IP normally attempt to resolve names using the following sources:

1. BIND/DNS (named)
2. Network Information Service (NIS)
3. Local /etc/hosts file

The default order described above can be overridden by using the NSORDER environment variable (set it in the /etc/environment file), for example NSORDER=local,bind, or in AIX 4, the /etc/netsvc.conf file, adding a line such as this:

```
host = local,bind
```

This example shows how to find our local system in the DNS (it is more detailed if the debugging flag -d is turned on as well). For example:

```
# nslookup $(hostname -s)
Server:  riscserver.itso.ibm.com
Address:  9.12.0.30

Non-authoritative answer:
Name:    sp6en0.msc.itso.ibm.com
Address:  192.168.6.160
```

This could also be done with the `host` command, using the old style, then the new style, and finally the new style with the verbose flag turned on (more detailed information is shown if the debugging flag -d is turned on as well). For example:

```
# host $(hostname -s)
sp6en0 is 192.168.6.160,  Aliases:   sp6en0.msc.itso.ibm.com, cws

# host -n $(hostname -s)
sp6en0.msc.itso.ibm.com has address 192.168.6.160

# host -n -v $(hostname -s)
Trying domain "msc.itso.ibm.com"
rcode = 0 (Success), ancount=1
The following answer is not authoritative:
sp6en0.msc.itso.ibm.com 9999999 IN      A       192.168.6.160
```

Note

The host command uses a cache while the `nslookup` command always talks to the name server.

This example queries the DNS for nameservers (type=NS) for the specified host (only the domain could also be supplied):

```
# nslookup -querytype=NS  www.ibm.com
Server:  riscserver.itso.ibm.com
Address:  9.12.0.30

Non-authoritative answer:
www.ibm.com     nameserver = ns2.nyc.ibm.com
www.ibm.com     nameserver = ns.nyc.ibm.com

Authoritative answers can be found from:
ns2.nyc.ibm.com internet address = 9.242.119.176
ns.nyc.ibm.com  internet address = 9.38.112.84
```

The following example uses the new style of the `host` command to make a similar query (note that we have verbose mode switched on):

```
# host -n -v -t NS  www.ibm.com
Trying null domain
rcode = 0 (Success), ancount=2
The following answer is not authoritative:
www.ibm.com     23021 IN       NS      ns.nyc.ibm.com
www.ibm.com     23021 IN       NS      ns2.nyc.ibm.com
Additional information:
ns.nyc.ibm.com  42747 IN       A       9.38.112.84
ns2.nyc.ibm.com 42747 IN       A       9.242.119.176
```

11.1.21 ntpq

The `ntpq` command queries Network Time Protocol (NTP) servers about their current NTP state. It runs either in interactive mode or by using command line arguments. The `ntpq` command can, among other things, also obtain and print a list of peers in a common format by sending multiple queries to the server. For example:

```
# ntpq -c peers
     remote           refid      st t when poll reach   delay   offset
==============================================================================
*LOCAL(2)        LOCAL(2)         3 l   23   64  377     0.00    0.000
```

11.1.22 ntptrace

The `ntptrace` command traces a chain of Network Time Protocol (NTP) hosts back to their master time source. The `ntptrace` command determines where a given NTP server gets its time, and follows the chain of NTP servers back to their master time source.

The following example shows a `ntptrace` that failed due to time-out

```
# ntptrace
loopback: stratum 4, offset 0.000812, synch distance 0.01044
127.127.1.2:    *Timeout*
```

11.1.23 Performance Diagnostic Tool (PDT)

The PDT package attempts to identify performance problems automatically by collecting and integrating a wide range of performance, configuration, and availability data. The data is regularly evaluated to identify and anticipate common performance problems. PDT assesses the current state of a system and tracks changes in workload and performance.

PDT data collection and reporting are easily enabled, and then no further administrator activity is required.

Important

If no other structured way is used to monitor system performance, always enable PDT, and archive the reports.

While many common system performance problems are of a specific nature, PDT also attempts to apply some general concepts of well-performing systems to its search for problems. Some of these concepts are as follows:

- Balanced Use of Resources
- Operation within Bounds
- Identified Workload Trends
- Error-Free Operation
- Changes Investigated
- Appropriate Setting of System Parameters

11.1.23.1 Start and configure PDT

To start PDT, run the following command and use the menu driven configuration program:

```
# /usr/sbin/perf/diag_tool/pdt_config
```

When you run it, follow the menus. The next example is taken from the main menu:

```
________________PDT customization menu__________________

1) show current  PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable       PDT reporting
4) modify/enable PDT collection
5) disable       PDT collection
6) de-install    PDT
7) exit pdt_config
Please enter a number:
```

First, check the current setting by selecting 1, as shown in the following example:

```
current PDT report recipient and severity level
root 3

______________PDT customization menu________________

1) show current  PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable       PDT reporting
4) modify/enable PDT collection
5) disable       PDT collection
6) de-install    PDT
7) exit pdt_config
Please enter a number:
```

This states *level 3* reports are to be made and sent to the *root user* on the local system. To check if root has a mail alias defined, run the following command:

```
# grep ^root /etc/aliases
```

If nothing is returned, the mail should be delivered to the local node. If however, it returns something formatted, as the following example shows:

```
# grep ^root /etc/aliases
root:reports@night4n37.msc.itso.ibm.com,"|/usr/bin/cat >>/tmp/log"
```

It is routed to another user and/or node, in this case the user (or another alias) `reports` on the node night4n37, and in this case, the mail will also be appended to the /tmp/log file.

By default, Driver_ reports are generated with severity level 1 with only the most serious problems identified. Severity levels 2 and 3 are more detailed and can, as in this case, be enabled. Also, by default, the reports are mailed to the adm user, but, as in this case, it is changed to root (it can also be changed so that it is not sent at all).

The configuration program will update the adm users crontab file. Check the changes made by using the `cronadm` command, as in the following example:

```
# cronadm cron -l adm|grep diag_tool
0  9 * * 1-5   /usr/sbin/perf/diag_tool/Driver_ daily
0 10 * * 1-5   /usr/sbin/perf/diag_tool/Driver_ daily2
0 21 * * 6     /usr/sbin/perf/diag_tool/Driver_ offweekly
```

You can also use `grep` on the crontab file, as shown here:

```
# grep diag_tool /var/spool/cron/crontabs/adm
0  9 * * 1-5   /usr/sbin/perf/diag_tool/Driver_ daily
0 10 * * 1-5   /usr/sbin/perf/diag_tool/Driver_ daily2
0 21 * * 6     /usr/sbin/perf/diag_tool/Driver_ offweekly
```

The daily parameter makes the Driver_ program collect data and store it in the /var/perf/tmp directory. The programs that do the actual collecting are specified in the /var/perf/cfg/diag_tool/.collection.control file. These programs are also located in the /usr/sbin/perf/diag_tool directory.

The daily2 parameter makes Driver_ create a report from the /var/perf/tmp data files and e-mails it to the recipient specified in the /var/perf/cfg/diag_tool/.reporting.list file. The PDT_REPORT is the formatted version and the .SM_RAW_REPORT is the unformatted report file.

Reports generated by PDT

The following is a short extract from the PDT_REPORT file:

```
Performance Diagnostic Facility 1.0

 Report printed: Thu Oct 12 12:04:35 2000

 Host name: sp6en0
 Range of analysis includes measurements
 from: Hour 11 on Thursday, October 12th, 2000
 to: Hour 11 on Thursday, October 12th, 2000
<<< lines omitted >>>
-------------------- Summary -------------------------
  This is a severity level 3 report
  No further details available at severity levels > 3
```

The raw information from the .SM_RAW_REPORT file that is used for creating the PDT_REPORT file follows:

```
H 1  | Performance Diagnostic Facility 1.0
H 1  |

H 1  | Report printed: Thu Oct 12 12:04:35 2000

H 1  |

H 1  | Host name: sp6en0

H 1  | Range of analysis includes measurements

H 1  |    from: Hour 11 on Thursday, October 12th, 2000

H 1  |    to: Hour 11 on Thursday, October 12th, 2000
H 1  |

<<< lines omitted >>>
```

The PDT_REPORT, at level 3, will have the following report sections:

- Alerts
- Upward Trends
- Downward Trends
- System Health
- Other
- Summary

It will also have subsections such as the following:

- I/O Configuration
- Paging Configuration
- I/O Balance
- Processes
- File Systems
- Virtual Memory

The following script shows you how to extract report subsections from the PDT_REPORT file; in this example, it displays all subsections in turn:

```
#!/bin/ksh

set -A tab  "I/O CONFIGURATION" "PAGING CONFIGURATION" "I/O BALANCE" \
            "PROCESSES" "FILE SYSTEMS" "VIRTUAL MEMORY"

for string in "${tab[@]}";do
    grep -p "$string" /var/perf/tmp/PDT_*
done
```

As an alternative to using the periodic report, any user can request a current report from the existing data by executing /usr/sbin/perf/diag_tool/pdt_report # (where the # sign is the severity number 1 to 3). The report is produced with the given severity (if none is provided, it defaults to 1) and it is written to standard output. Generating a report in this way does not cause any change to the /var/perf/tmp/PDT_REPORT files.

11.1.23.2 Editing configuration files

There are some configuration files for PDT that need to be edited to better reflect the needs of a specific system.

Files and directories

PDT analyzes files and directories for systematic growth in size. It examines only those files and directories listed in the file /var/perf/cfg/diag_tool/.files. The format of the .files file is one file or directory name per line. The default content of this file is as follows:

```
/usr/adm/wtmp
/var/spool/qdaemon/
/var/adm/ras/
/tmp/
```

You can use an editor[2] to modify this file to track files and directories that are important to your system.

Hosts

PDT tracks the average ECHO_REQUEST delay to hosts whose names are listed in the /var/perf/cfg/diag_tool/.nodes file. This file is not shipped with PDT (which means that no host analysis is performed by default), but they may be created by the administrator. The format of the .nodes file is one host name per line in the file. /etc/resolv.conf points to this IP address. For example:

```
# awk '/nameserver/{print $2}' /etc/resolv.conf
9.12.0.30
```

For example, to monitor the nameserver used by the local node, the .nodes file would contain the following line:

```
9.12.0.30
```

Thresholds

The file /var/perf/cfg/diag_tool/.thresholds contains the thresholds used in analysis and reporting. These thresholds, listed below, have an effect on PDT report organization and content. The following is the content of the default file:

```
# grep -v ^# .thresholds
DISK_STORAGE_BALANCE 800
PAGING_SPACE_BALANCE 4
NUMBER_OF_BALANCE 1
MIN_UTIL 3
FS_UTIL_LIMIT 90
MEMORY_FACTOR .9
TREND_THRESHOLD .01
EVENT_HORIZON 30
```

[2] Do not use vi; just append filenames with: `print filename >> .files`

The settings in the preceding content listing are the default values, and the following are the basic description for each threshold:

DISK_STORAGE_ BALANCE	The SCSI controllers with the largest and smallest disk storage are identified. This is a static size, not the amount allocated or free.The default value is 800. Any integer value between 0 and 10000 is valid.
PAGING_SPACE_ BALANCE	The paging spaces having the largest and the smallest areas are identified. The default value is 4. Any integer value between 0 and 100 is accepted. This threshold is presently not used in analysis and reporting.
NUMBER_OF_ BALANCE	The SCSI controllers having the largest and the least number of disks attached are identified.The default value is 1. It can be set to any integer value in the range of 0 to 10000.
MIN_UTIL	Applies to process utilization. Changes in the top three CPU consumers are only reported if the new process had a utilization in excess of MIN_UTIL. The default value is 3. Any integer value from 0 to 100 is valid.
FS_UTIL_LIMIT	Applies to journaled file system utilization. Any integer value between 0 and 100 is accepted.
MEMORY_FACTOR	The objective is to determine if the total amount of memory is adequately backed up by paging space. The formula is based on experience and actually compares MEMORY_FACTOR * memory with the average used paging space. The current default is 0.9. By decreasing this number, a warning is produced more frequently. Increasing this number eliminates the message altogether. It can be set anywhere between .001 and 100.
TREND_THRESHOLD	Used in all trending assessments. It is applied after a linear regression is performed on all available historical data. This technique basically draws the best line among the points. The slope of the fitted line must exceed the last_value * TREND_THRESHOLD. The objective is to try to ensure that a trend, however strong its statistical significance, has some practical significance. The threshold can be set anywhere between 0.00001 and 100000.

EVENT_HORIZON	Used also in trending assessments. For example, in the case of file systems, if there is a significant (both statistical and practical) trend, the time until the file system is 100 percent full is estimated. The default value is 30, and it can be any integer value between 0 and 100000.

11.1.24 ping

The `ping` command sends an Internet Control Message Protocol (ICMP) ECHO_REQUEST to obtain an ICMP ECHO_RESPONSE, from a host or gateway. The `ping` command is useful for:

- Determining the status of the network and foreign hosts.
- Tracking and isolating hardware and software problems.
- Testing, measuring, and managing networks.

If the host is operational and on the network, it responds to the ECHO_REQUEST. Each default ECHO_REQUEST contains an Internet Protocol (IP) and ICMP header, followed by a timeval structure, and enough bytes to fill out the packet (the default is 56 bytes).

The following example is similar to the one used with `traceroute` and shows the route to and from a remote node:

```
# ping -c 1 -R pollux.ibm.com
PING pollux.ibm.com: (9.46.1.2): 56 data bytes
64 bytes from 9.46.1.2: icmp_seq=0 ttl=252 time=60 ms
RR:     9.32.41.42
        9.32.44.1
        9.32.1.93
        9.46.1.1
        pollux.cbe.ibm.com (9.46.1.2)
        9.32.1.94
        9.32.44.3
        9.32.41.41

----pollux.ibm.com PING Statistics----
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 60/60/60 ms
```

But the real use of `ping` is to quickly check if systems are available on the network or not, as the following example shows (by sending 1 default packet):

```
# ping -c 1 w3.ibm.com
PING w3ibm.southbury.ibm.com: (9.45.3.170): 56 data bytes
64 bytes from 9.45.3.170: icmp_seq=0 ttl=249 time=35 ms

----w3ibm.southbury.ibm.com PING Statistics----
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 35/35/35 ms
```

But ping can also be used to saturate networks if used inappropriately, as the following example shows. This command sends as many packets it can to the remote host with 8184 byte per packet (For every ECHO_REQUEST sent, a . (period) is printed, while for every ECHO_REPLY received, a backspace is printed). This particular example is run over a SP Switch2 (note that ping is not appropriate to use in order to measure network throughput):

```
# ping -f -d night4s33_00 8184
.......................................................................
<<< lines omitted >>>
......................................................^C
----night4s33_00.ppd.pok.ibm.com PING Statistics----
923590 packets transmitted, 470184 packets received, 49% packet loss
round-trip min/avg/max = 0/1/108 ms
```

The following is a small sample of the netstat data collected during the run of two simultaneous ping commands, as described above, from one system to the monitoring one:

```
# netstat -I css0 5
input    (css0)      output            input    (Total)     output
 packets   errs  packets   errs colls  packets   errs  packets   errs colls
 2097782      0  2013842      0     0  4951165      0  6831574      0     0
   33145      0    33146      0     0    33165      0    33169      0     0
   32987      0    32984      0     0    32999      0    32993      0     0
   33009      0    33009      0     0    33019      0    33019      0     0
   33099      0    33098      0     0    33110      0    33111      0     0
   32873      0    32872      0     0    32882      0    32882      0     0
   32744      0    32743      0     0    32756      0    32752      0     0
<<< lines omitted >>>
```

Another good use of ping is to send packets to remote hosts, to test if the packets will be fragmented on the way. This requires tracing the network with tcpdump or iptrace during the transfer.

11.1.25 pprof

The pprof command reports CPU usage of all kernel threads over a period of time and saves the data to files. Table 34 shows the report files generated by pprof.

Table 34. Files produced by pprof

File	Content description
pprof.cpu	Lists all kernel level threads sorted by actual CPU time.
pprof.start	Lists all kernel threads sorted by start time.

File	Content description
pprof.namecpu	Lists information about each type of kernel thread (all executable with the same name).
pprof.famind	Lists all processes grouped by families (processes with a common ancestor).
pprof.famcpu	Lists the information for all families (processes with a common ancestor).

The following example shows a good usage of `pprof` to monitor processes for a specified duration (in this example, one minute):

```
# pprof 60 -n
Sat Oct  7 14:54:11 2000
System: AIX sp6en0 Node: 4 Machine: 000504936700

61.306 secs in measured interval

*** PPROF COMPLETED ***
# head -20 pprof.namecpu

                    Pprof  PROCESS  NAME  Report

                    Sorted  by  CPU  Time

                    From: Sat Oct  7 15:06:54 2000
                    To:   Sat Oct  7 15:07:56 2000


               Pname #ofThreads  CPU_Time        %
            ======== ==========  ========  ========
                 ksh          1    56.635    92.440
                 gil          4     1.769     2.887
      sr/dt/bin/dtexec        2     0.518     0.845
           dtsession          2     0.498     0.813
             hardmon          3     0.442     0.721
               hatsd          4     0.413     0.674
                sdrd          4     0.144     0.235
              dtfile          4     0.132     0.215
```

11.1.26 ps

The `ps` command is a very flexible tool for identifying the programs that are running on the system and the resources they are using. It displays statistics and status information about processes on the system, such as process or thread ID, I/O activity, CPU and memory utilization.

Note

The `ps` command can only show information on running processes. If you have processes that are starting and terminating before you can make a snapshot with `ps`, process accounting must be turned on, so that information about processes are archived after their termination.

11.1.26.1 Monitoring threads

If the -m flag is used, it also gives the status of associated kernel threads. To display extra thread-related columns, you must use the -o THREAD flag in conjunction with the -m flag.

With Deferred Page Space Allocation (DPSA), the paging space may not get touched. This means that with DPSA on, you could see smaller values for SIZE and SZ when using `ps`.

The following example examines the nfsd daemon. First, we check it when it uses the default number of eight daemon threads:

```
# lssrc -s nfsd
Subsystem        Group            PID     Status
 nfsd            nfs              4478    active

# ps -mo THREAD -p 4478
    USER   PID  PPID    TID ST  CP PRI SC    WCHAN        F     TT BND COMMAND
    root  4478  9032      - A    0  60  3        *   240001      -   - /usr/sbin
/nfsd
       -     -     -   8529 S    0  60  1 50044608     1400      -   - -
       -     -     -  10857 S    0  60  1 5278cd98      400      -   - -
       -     -     -  11095 S    0  60  1   36c8bc     1400      -   - -
```

We now change the maximum number of daemon threads allowed and then check the new instance of nfsd and look at how many have been started:

```
# chnfs -n 4096
0513-044 The nfsd Subsystem was requested to stop.
0513-077 Subsystem has been changed.
0513-059 The nfsd Subsystem has been started. Subsystem PID is 4480.

# ps -mo THREAD -p 4480
    USER   PID  PPID    TID ST  CP PRI SC    WCHAN        F     TT BND COMMAND
    root  4480  9032      - A    0  60  3        *   240001      -   - /usr/sbin
/nfsd
       -     -     -   8531 S    0  60  1 50028608     1400      -   - -
       -     -     -  10859 S    0  60  1 5275b018      400      -   - -
       -     -     -  11097 S    0  60  1   36c8bc     1400      -   - -
```

After running the following transfer test from a client:

```
# dd if=/dev/zero cbs=65536 count=1000000 of=zero.out
```

Checking the nfsd process on the server shows a lot of zombie threads:

```
# ps -mo THREAD -p 6212
    USER   PID  PPID    TID ST  CP PRI SC    WCHAN        F     TT BND COMMAND
    root  6212  9032      - A    0  60 16        *   240001      -   - /usr/sbin
/nfsd
       -     -     -  10863 Z    0  60  1        -     1001      -   - -
       -     -     -  11371 S    0  60  1   36c7e8     1400      -   - -
       -     -     -  11625 S    0  60  1 500014d8      400      -   - -
       -     -     -  22777 Z    0  60  1        -     1001      -   - -
       -     -     -  26267 Z    0  60  1        -     1001      -   - -
       -     -     -  27007 S    0  60  1 50028708     1400      -   - -
       -     -     -  27163 Z    0  60  1        -     1001      -   - -
       -     -     -  28145 Z    0  60  1        -     1001      -   - -
       -     -     -  28391 Z    0  60  1        -     1001      -   - -
       -     -     -  28471 Z    0  60  1        -     1001      -   - -
       -     -     -  28793 Z    0  60  1        -     1001      -   - -
       -     -     -  29119 Z    0  60  1        -     1001      -   - -
       -     -     -  29275 Z    0  60  1        -     1001      -   - -
       -     -     -  29673 Z    0  60  1        -     1001      -   - -
       -     -     -  29929 Z    0  60  1        -     1001      -   - -
```

11.1.26.2 Monitoring processes

Here are some useful option combinations for the `ps` command.

gvc

The gvc options instructs `ps` to display:

- All processes
- The PGIN, SIZE, RSS, LIM, TSIZ, TRS, %CPU, %MEM fields
- The command name, as stored internally in the system for purposes of accounting, rather than the command parameters, which are kept in the process address space

The following is a sample output with explanation for the fields:

```
# ps gvc
PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
     0      - A     0:27    7   268 13932    xx     0 13664  0.0  1.0 swapper
     1      - A     0:05  131   908   996 32768    25    36  0.0  0.0 init
   516      - A  2758:33    0   264 13928    xx     0 13664 24.9  1.0 kproc
   774      - A  2745:00    0   264 13928    xx     0 13664 24.8  1.0 kproc
  1032      - A  2750:55    0   264 13928    xx     0 13664 24.9  1.0 kproc
  1290      - A  2757:10    0   264 13928    xx     0 13664 24.9  1.0 kproc
<<< lines omitted >>>
 21028  pts/1 T     0:09    0   276   340 32768    22    32  0.0  0.0 ping
 22150      - A     0:00    0   328   428 32768    25    44  0.0  0.0 inetd
 22494  pts/0 A     0:00    0   412   504 32768    52    64  0.0  0.0 ps
```

The columns are explained as follows:

TTY The controlling workstation for the process.

STAT Contains the state of the process: 0 - Nonexistent, A - Active, I - Intermediate, Z - Canceled, T - Stopped, K - Available kernel process

TIME The total execution time for the process.

PGIN The number of disk I/Os resulting from references by the process to pages not loaded.

SIZE The virtual size of the data section of the process (in 1 KB units).

RSS The real-memory (resident set) size of the process (in 1 KB units).

LIM The soft limit on memory used. If no limit has been specified, then it is shown as xx. If the limit is set to the system limit (unlimited), a value of UNLIM is displayed.

TSIZ The size of text (shared-program) image.

TRS The size of resident-set (real memory) of text.

%CPU The percentage of time the process has used the CPU since the process started. The value is computed by dividing the time the process uses the CPU by the elapsed time of the process. In a multi-processor environment, the value is further divided by the number of available CPUs since several threads in the same process can run on different CPUs at the same time. (Because the time base over which this data is computed varies, the sum of all %CPU fields can exceed 100%.)

%MEM The percentage of real memory used by this process.

Other optional combinations

Other useful options to the `ps` command are shown with their headers below:

```
# ps vax|head -1;
PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND

# ps lax|head -1
F S      UID   PID  PPID   C PRI NI ADDR  SZ  RSS   WCHAN    TTY  TIME CMD

# ps -elk|head -1
F S      UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
```

The following example shows how to use `ps` together with `sort` and `head` to create sorted output of the highest consumers. In this case, we sort on column 4, which is TIME:

```
# LANG=C ps gvc|sort -rk4|head -10
   PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
   516      - A    2801:04    0   264 13928    xx     0 13664 24.9  1.0 kproc
  1290      - A    2799:40    0   264 13928    xx     0 13664 24.9  1.0 kproc
  1032      - A    2793:24    0   264 13928    xx     0 13664 24.9  1.0 kproc
   774      - A    2787:30    0   264 13928    xx     0 13664 24.8  1.0 kproc
 11248      - A     5:51  299  8132  8932 32768   621   684  0.1  1.0 hatsd
  2064      - A     2:30    0   320 13984    xx     0 13664  0.0  1.0 kproc
  3196      - A     1:18    4   416   440    xx     2     4  0.0  0.0 syncd
 18090      - A     1:10  344  1376  2512 32768  1884  1520  0.0  0.0 i4llmd
  9050      - A     1:06 5367   416   464 32768     4     8  0.0  0.0 nfsd
```

If we wanted the results sorted on another column, we would just change the number 4 for the `sort` command.

11.1.27 pstat

The `pstat` command is a non-interactive form of the `crash` command. `pstat` interprets the contents of the various system tables and writes it to standard output. You must have root user or system group authority to run the `pstat` command.

The following example shows how to examine all runnable threads:

```
# pstat -P
THREAD TABLE:

SLT ST    TID      PID    CPUID  POLICY PRI CPU    EVENT  PROCNAME     FLAGS
  2 r     205      204        0    FIFO  7f  78              wait
        t_flags:  sig_avail funnel kthread
  3 r     307      306        1    FIFO  7f  78              wait
        t_flags:  sig_avail funnel kthread
  4 r     409      408        2    FIFO  7f  78              wait
        t_flags:  sig_avail funnel kthread
  5 r     50b      50a        3    FIFO  7f  78              wait
        t_flags:  sig_avail funnel kthread
109 r    6da5     4d38  unbound   other  58  38             pstat
        t_flags:
```

It is possible to use `pstat` to monitor paging space usage; the USED PAGES column shows pages that are allocated on a paging space:

```
# pstat -s
PAGE SPACE:

          USED PAGES   FREE PAGES
              453      2358843
```

To display the status of processors in the node, use the -S flag as in the following example (with 12 CPUs):

```
# pstat -S
STATUS OF PROCESSORS:

CPU     TID  TSLOT     PID  PSLOT  PROC_NAME
  0  25e5db   9701   18544    389       no  pstat
  1     307      3     306      3       no  wait
  2     e1d     14     e1c     14       no  lrud
  3  422879  16936   4268c   1062       no  oracle
  4  6e899d  28297   1e496    484       no  oracle
  5     70f      7     70e      7       no  wait
  6  55f4c3  22004   3a16a    929       no  oracle
  7     913      9     912      9       no  wait
  8     a15     10     a14     10       no  wait
  9     b17     11     b16     11       no  wait
 10  25e5db   9701   18544    389       no  pstat
 11     d1b     13     d1a     13       no  wait
```

Since `pstat` is capable of extracting information from the kernel memory space, we can also retrieve information on running process statistics and resource limits and usage. The following example shows how to extract the resource limits that concerns the running nfsd process. These limits can be set for users in /etc/security/limits but also at runtime, and if we would like to check what limits were imposed on a vital server process, this is the easiest way to do it (but the hex values need conversion):

```
# pstat -u $(pstat -a|awk '/nfsd/{print $1}')|grep -p '^RESOURCE LIMITS'
RESOURCE LIMITS AND COUNTERS
    ior:0x00000000_00000000  iow:0x00000000_00000000  ioch:0x00000000_0000824c
    text:0x00000000_00000e58  data:0x00000000_00015000  stk:0x02000000
    max data:0x08000000  max stk:0x02000000  max file(blks):0xffffffff
    *tstart:0x00000000_10000100  sdsize:0x00000000
    *datastart:0x00000000_20000000  *stkstart0x00000000_2ff23000
    soft core dump:0x3ffffe00  hard core dump:0x7fffffff
    soft rss:0x02000000  hard rss:0x7fffffff
    cpu soft:0x7fffffff  cpu hard:0x7fffffff
    hard ulimit:0x7fffffff
    minflt:0x00000000_00000000   majflt:0x00000000_00000000
```

11.1.28 sar

The `sar` command reports either system wide (global among all processors) CPU statistics (which are calculated as averages for values expressed as percentages, and as sums otherwise), or it reports statistics for each individual processor. Therefore, this command is particularly useful on SMP systems.

When used with the -P flag, the information provided for each specified processor; otherwise, it is provided system wide. Processors can be specified individually or together, or the keyword ALL can be used.

Processor utilization

Our next example reports CPU utilization per processor or system wide statistics:

```
# sar -P ALL 5 2

AIX night4n33 3 4 006014524C00    10/11/00

20:14:19 cpu    %usr    %sys    %wio   %idle
20:14:24  0        0      14       0      86
          1       90      10       0       0
          2        0      15       0      85
          3        0      11       0      89
          -       22      12       0      65
<<< lines omitted >>>
Average   0        0      12       0      88
          1       71      11       0      18
          2        0      12       0      88
          3        0      12       0      88
          -       18      12       0      70
```

The following values are displayed:

%idle	Reports the percentage of time the CPU or CPUs were idle with no outstanding disk I/O requests.
%sys	Reports the percentage of time the CPU or CPUs spent in execution at the system (or kernel) level.
%usr	Reports the percentage of time the CPU or CPUs spent in execution at the user (or application) level.
%wio	Reports the percentage of time the CPU(s) were idle during which the system had outstanding disk/NFS I/O request(s).

The following example illustrates that individual CPUs can be monitored separately:

```
# sar -P 3  5 3

AIX night4n33 3 4 006014524C00    10/11/00

20:18:24 cpu    %usr    %sys    %wio   %idle
20:18:29  3       93       7       0       0
20:18:34  3       90      10       0       0
20:18:39  3       90      10       0       0

Average   3       91       9       0       0
```

Block I/O

The following example reports activity for each block device:

```
# sar -d  5 1

AIX night4n37 3 4 006015614C00    10/11/00

21:34:38     device    %busy    avque    r+w/s   blks/s   avwait   avserv

21:34:43     hdisk3      0      0.0        0        5      0.0      0.0
             hdisk4      0      0.0        0        4      0.0      0.0
             hdisk2      0      0.0        0       24      0.0      0.0
             hdisk5      0      0.0        0        4      0.0      0.0
             hdisk0      0      0.0        0       12      0.0      0.0
             hdisk1      1      0.0        1       33      0.0      0.0
             hdisk6      0      0.0        2       58      0.0      0.0
```

The activity data reported is:

%busy	Reports the portion of time the device was busy servicing a transfer request.
avque	Average number of requests outstanding during that time.
r+w/s	Reports the number of read/write transfers from or to a device.
blks	Number of bytes transferred in 512-byte units.

Run queue

The following example reports queue statistics:

```
# sar  -q  5 2

AIX night4n37 3 4 006015614C00    10/11/00

21:25:04 runq-sz %runocc swpq-sz %swpocc
21:25:09                    1.0     100
21:25:14     1.0      20    1.0     100

Average      1.0      10    1.0     100
```

The activity data reported is:

runq-sz	Reports the average number of kernel threads in the run queue.
%runocc	Reports the percentage of the time the run queue is occupied.
swpq-sz	Reports the average number of kernel threads waiting to be paged in.

%swpocc	Reports the percentage of the time the swap queue is occupied.

A blank value in any column indicates that the associated queue is empty.

Context switching

The following example reports system switching activity:

```
# sar -P ALL -w  5 2

AIX night4n37 3 4 006015614C00    10/11/00

21:17:06 cpu cswch/s
21:17:11  0       29
          1       14
          2       21
          3       15
          -       78
<<< lines omitted >>>
Average   0       28
          1       13
          2       21
          3       15
          -       77
```

The activity data reported is:

cswch/s	Reports the number of context switches per second.

System calls

The following example reports system calls per processor:

```
# sar -P ALL -c  5 2

AIX night4n37 3 4 006015614C00    10/11/00

21:08:04 cpu scall/s sread/s swrit/s  fork/s  exec/s rchar/s wchar/s
21:08:09  0       48      12       0    0.00    0.00       0       0
          1       14       0       0    0.00    0.00       0       0
          2       24       3       1    0.00    0.00     262      23
          3       79       4       3    0.00    0.00     754     262
          -      510     110       4    0.00    0.00  368866     288
<<< lines omitted >>>
Average   0       49      12       0    0.00    0.00       0       0
          1       14       0       0    0.00    0.00       0       0
          2       23       3       1    0.00    0.00     262      46
          3       52       4       3    0.00    0.00     754     262
          -      310      64       4    0.00    0.00  184757     310
```

The activity data reported are:

exec/s, fork/s	Reports the total number of exec and fork system calls.
sread/s, swrit/s	Reports the total number of read/write system calls.
rchar/s, wchar/s	Reports the total number of characters transferred by read/write system calls.
scall/s	Reports the total number of system calls.

11.1.29 spmon

The `spmon` command can both operate on the SP system controls and monitor SP system activity. In its latter capacity, it is very useful, because it will quickly show vital points about the availability of the SP complex, as is shown in the following example for the fourth frame (Frame 4):

```
# spmon -G -d|grep -p 'Frame 4'
----------------------------------- Frame 4 ----------------------------------
                      Host     Switch   Key     Env   Front Panel      LCD/LED
Slot Node Type  Power Responds Responds Switch  Error LCD/LED          Flashes
---- ---- ----- ----- -------- -------- ------- ----- ---------------- -------
 12    60  wide   on     no       yes     N/A     no LCDs are blank        no
 14    62  wide   on     no       no      N/A     no LCDs are blank        no
```

11.1.30 statvsd

The `statvsd` command displays Virtual Shared Disk (VSD) device driver statistics of a node.

The following shows the default display when running `statvsd`. Since most VSD logical resources are self-tuned, monitoring for tuning of logical resources are not as necessary as before. However, Buddy Buffers can still benefit from monitoring and adjustment. Monitor the “requests queued waiting for a buddy buffer”; with a non-zero value, it indicates that more Buddy Buffers are needed (the value is accumulated). The field “buddy buffer wait_queue size“ is the current queue size

.

```
# statvsd
VSD driver (vsdd): KLAPI interface:     PSSP Version:3  Release: 2

         9 vsd parallelism
     61440 vsd max IP message size
         0 requests queued waiting for a request block
         0 requests queued waiting for a pbuf
         0 requests queued waiting for a cache block
         0 requests queued waiting for a buddy buffer
       0.0 buddy buffer wait_queue size
         0 rejected requests
         0 rejected responses
         0 rejected no buddy buffer
         0 rejected merge timeout.
         0 requests rework
         0 indirect I/O
         0 64byte unaligned reads.
         0 comm. buf pool shortage
         0 DMA space shortage
         0 timeouts
retries: 0 0 0 0 0 0 0 0 0
         0 total retries
Non-zero Sequence numbers
 node#       expected       outgoing   outcast?       Incarnation: 0
     1              5              0
     3            368              0
     9             12              0
    11             12              0
    12             12              0
    13             12              0
    14             12              0

 2 Nodes Up with zero sequence numbers: 5 7
```

11.1.31 svmon

The svmon command is a virtual memory monitor. When invoked, it captures a “snapshot” of the current contents of both real and virtual memory, and summarizes the contents. svmon is useful to determine which processes, or, alternatively, which segments are consuming the most real memory. svmon uses *virtual* counters from VMM for statistical analysis (these might not always be current).

Note

The displayed information does not constitute a true snapshot of memory, because the svmon command runs in the foreground as a normal user process and could be interrupted while collecting data. Be careful when running svmon snapshots on very busy systems with many processes.

The following shows the default display when running `svmon`; to monitor on a continuing basis, use the -i option with an interval number and a count number (`svmon -i 5 12`, for example, takes a snapshot every 5 seconds 12 times):

```
# svmon

              size      inuse       free        pin    virtual
memory      524277     430150      89373     524277      32328
pg space   1310720        439

              work       pers       clnt
pin          36613        156          0
in use       50477      13495     366178
```

In the first part of the output, what we usually are most interested in are the number of real memory pages that are inuse and free, as shown on the memory line. The number of pg space pages that are inuse show how many pages are actually in use on the paging space(s). The last line in use shows the division in memory to different segment types. See Section 6.1, “Memory” on page 115.

Since paging is usually to be avoided, use `svmon` (or `lsps`; see Section 11.1.13, “lsps” on page 265) to monitor paging space usage.

The following example shows how to use `svmon` to check all memory allocations for a specified process:

```
# svmon -C nfsd|sed '/^$/d'
===============================================================================
Command nfsd             Inuse      Pin     Pgsp  Virtual
                          2207        9       74     3513
...............................................................................
SYSTEM segments          Inuse      Pin     Pgsp  Virtual
                            48        8       13       49
  Vsid      Esid Type Description           Inuse   Pin Pgsp Virtual Addr Range
  5245         - work                          42     2   13      42  0..49746
  e00e         0 work kernel shadow             6     6    0       7  0..12
...............................................................................
EXCLUSIVE segments       Inuse      Pin     Pgsp  Virtual
                            50        1       35      103
  Vsid      Esid Type Description           Inuse   Pin Pgsp Virtual Addr Range
  6246         2 work process private          50     1   19      87  0..20 :
                                                                      65308..65535
  b24b         f work shared library data       0     0   16      16  0..422
  8248         1 pers code,/dev/hd2:306         0     0    -       -  0..1
...............................................................................
SHARED segments          Inuse      Pin     Pgsp  Virtual
                          2109        0       26     3361
  Vsid      Esid Type Description           Inuse   Pin Pgsp Virtual Addr Range
 13013         d work shared library text    2109     0   26    3361  0..65535
```

The following command displays the memory usage statistics for the top process (1 was only selected since the output is quite extensive from this command). If you do not specify a number, it will display all the processes currently running in this system, and the -i option lets you specify interval and report count. By specifying a process ID, after -P, a specific process can be examined.

```
# svmon -P -z -t 1

-------------------------------------------------------------------------------
     Pid Command        Inuse      Pin     Pgsp  Virtual   64-bit    Mthrd
   14192 bin             4100      165      899     5227        N        Y

  Vsid     Esid Type Description          Inuse   Pin Pgsp Virtual Addr Range
 13013        d work shared library text   2677     0   20  3375   0..65535
  b38b        2 work process private       1141     1  843  1730   0..1862 :
                                                                   65055..65535
 14494        1 pers code,/dev/hd2:121188   156   156    -     -   0..170
  c38c        f work shared library data     53     0   24    56   0..476
 18498        - work                         27     0    0    27   0..26
 1d4dd        - work                         19     2   12    30   0..49746
  a4aa        - pers /dev/hd9var:1246        15     0    -     -   0..62
  e00e        0 work kernel shadow            7     6    0     7   0..12
  f3cf        - pers /dev/hd2:59502           2     0    -     -   0..18
 1f4df        - work                          1     0    0     1   0..0
 1c4dc        - work                          1     0    0     1   0..0
  64c6        - pers /dev/hd9var:1268         1     0    -     -   0..0
Maximum memory allocated = 47888
```

As is shown in this example, `svmon` examines segment (b38b) that is used by the bin process above:

```
# svmon -D b38b -b -z

Segid: b38b
Type:  working
Address Range: 0..1862 : 65055..65535
Size of page space allocation: 998 pages (  3.9 Mb)
Virtual: 1730 frames ( 6.8 Mb)
Inuse: 885 frames ( 3.5 Mb)

           Page      Frame    Pin        Ref        Mod
          65339      50875      Y          Y          Y
          65340      18170      N          Y          Y
          65407      21053      N          N          Y
          65338      14246      N          N          Y
<<< lines omitted >>>
           1732      63150      N          Y          Y
           1733      63125      N          Y          Y
            499      52937      N          N          Y
Maximum memory allocated = 14976
```

When using the -S option, `svmon` sorts segments by memory usage and displays the memory-usage statistics for the top memory-usage segments:

```
# svmon -S -t 5

  Vsid     Esid Type Description           Inuse   Pin Pgsp Virtual Addr Range
 13013        - work                        2677     0   20  3375   0..65535
     0        - work kernel seg             1964  1290  774  3967   0..20698 :
                                                                    65423..65535
  f00f        - work misc kernel tables     1840     0  522  2154   0..15017 :
                                                                    63488..65535
 165b6        - pers /dev/lv00:2259         1451     0    -     -   0..1450
 10010        - work kernel pinned heap     1349   587   99  1439   0..65535
```

By using the -C option in a script, we can extract and display only the information we want to look at. In this case, the sum for each application split into non-shared, shared and total memory usage. For example:

```
#!/bin/ksh

tmpfile=/tmp/tmp$$
trap 'rm -f $tmpfile' EXIT

printf "%-20.20s %6s %6s %6s\n" \
    "Application" "Mem/Instance(KB)" "Mem/Shared(KB)" "Mem/total(KB)"
for p in $(ps -eF "%c"|tail +2|sort -u);do
    svmon -C $p >$tmpfile
    [[ $(grep -c 'command does not exist' $tmpfile) -gt 0 ]] && continue
    grep -p segments $tmpfile|
    awk 'BEGIN{ i=0 }
        { if ($1 ~/[0-9]/) tab[i++]=$1 }
        END{ printf "%-20.20s %12d   %12d  %12d\n",
                proc,(tab[1]),(tab[0]+tab[2]),(tab[0]+tab[1]+tab[2])
        }' proc=$p
done
```

But note that `svmon` sometimes fails to report on some commands while using the -C option. But here is a sample output from the preceding script:

```
Application          Mem/Instance(KB) Mem/Shared(KB) Mem/total(KB)
biod                          192           9052          9244
bootpd                        320           9052          9372
cron                          504           9052          9556
dpid2                          32           9052          9084
<<< lines omitted >>>
```

11.1.32 tcpdump

The `tcpdump` command prints out the headers of packets captured on a network interface that matches the specified selection criteria. Only Ethernet, Token-Ring, FDDI and loopback interfaces are supported. Access is controlled by the permissions on /dev/bpf0,1,2, and 3.

Timestamps

By default, all output lines are preceded by a timestamp. The timestamp is the current clock time in the form:

```
hh:mm:ss.frac
```

and is as accurate as the kernel's clock. The timestamp reflects the time the kernel first saw the packet. No attempt is made to account for the time lag between when the ethernet interface removed the packet from the wire and when the kernel serviced the new packet interrupt.

Turn off timestamp reporting with the -t parameter.

ARP

The following example shows how to use `tcpdump` to monitor ARP traffic on a network on a node:

```
# tcpdump -NI arp net 129.40.35
tcpdump: listening on en0
10:28:20.596472784 arp who-has night4n33 tell night4n37
10:28:20.596628482 arp reply night4n33 is-at 0:4:ac:ec:7:98
10:28:54.086867216 arp who-has night4n01 tell night4n09
10:28:54.094944743 arp who-has night4n17 tell night4n09
10:28:54.096532951 arp who-has night4n21 tell night4n09
10:28:54.127165486 arp who-has night4n29 tell night4n09
10:28:54.148348504 arp who-has night4n09 tell 129.40.35.116
10:28:55.029989360 arp who-has night4n09 tell net_8271
```

The two first time stamped lines is our host asking the network for the MAC address of host night4n33 and it got a reply from night4n33 that it has a MAC address of 0:4:ac:ec:7:98. The next six lines are requests from other hosts for other hosts MAC addresses, the node sending the ARP request is the host that wishes to be told about someones MAC address, and it is the host that has the IP address, that is queried, that is supposed to answer.

TCP

The next sample shows monitoring of all the start and end packets (the SYN and FIN packets) of each TCP conversation on the local node:.

```
# tcpdump -NIt \(tcp[13] \& 3 !=0 \)
tcpdump: listening on en0
night4n33.41819 > night4n37.telnet: S 2377895223:2377895223(0) win 65535 <mss 14
60,nop,wscale 1,nop,nop,timestamp 171568130 1728053248> [tos 0x10]
night4n37.telnet > night4n33.41819: S 3889971395:3889971395(0) ack 2377895224 wi
n 63712 <mss 1460,nop,wscale 0,nop,nop,timestamp 171566428 456780802>
```

The output above shows the start of a telnet session to our monitoring node from node night4n33 (we are night4n37). Note the request for *message* *segment size* of 1460. We see the reply to the TCP initiation in the following example:

```
night4n37.telnet > night4n33.41814: F 404724337:404724337(0) ack 3524856541 win
63712 <nop,nop,timestamp 171566426 3527011329>
night4n33.41814 > night4n37.telnet: F 1:1(0) ack 1 win 31856 <nop,nop,timestamp
171568129 507110746> [tos 0x10]
```

11.1.33 topas

The `topas` command reports selected statistics about activities on the local system. Next to many other statistics, it provides CPU utilization information (user, kernel, idle, and I/O wait) for either a selected number of CPUs or the average over all CPUs. `topas` also shows information on the number of context switches, system calls, reads, writes, forks, and execs. The process statistics shown include the amount of threads in run and wait queues, a selectable number of processes, and the average CPU utilization for each process displayed.

```
Topas Monitor for host:    night4n33           EVENTS/QUEUES    FILE/TTY
Thu Oct  5 16:25:24 2000   Interval:  2        Cswitch   24783  Readch  7652769
                                               Syscall   87706  Writech 7651225
Kernel   32.6   |#########                  |  Reads     43822  Rawin         0
User      9.3   |###                        |  Writes    43804  Ttyout      130
Wait     46.0   |##                         |  Forks         0  Igets         0
Idle     52.0   |###############            |  Execs       3.6  Namei         1
                                               Runqueue    2.4  Dirblk        0
Interf   935.9   473.4  5383.9    61.5  7874.4 Waitqueue   1.1
en0     7749.7   463.9  5258.4    60.3  7689.3
lo0        0.0     0.0     0.0     0.0     0.0 PAGING           MEMORY
                                               Faults     1840  Real,MB    2047
Disk    Busy%     KBPS     TPS KB-Read KB-Writ Steals        0  % Comp     10.0
hdisk1    0.0      0.0     0.0     0.0     0.0 PgspIn        0  % Noncomp  17.0
hdisk0    0.0      0.0     0.0     0.0     0.0 PgspOut       0  % Client   15.0
                                               PageIn     1830
dd        (26526) 64.0% PgSp: 0.0mb root       PageOut    1823  PAGING SPACE
dd        (30590) 60.5% PgSp: 0.0mb root       Sios       2057  Size,MB    5120
kbiod     (4692)  49.5% PgSp: 0.0mb root                        % Used      0.0
hatsd     (6538)   0.0% PgSp: 6.1mb root                        % Free     99.9
topas     (28352)  0.0% PgSp: 0.4mb root
ksh       (29942)  0.0% PgSp: 0.3mb root
fault_ser(13724)   0.0% PgSp: 2.1mb root          Press "h" for help screen.
gil       (2064)   0.0% PgSp: 0.0mb root          Press "q" to quit program.
ksh       (27796)  0.0% PgSp: 0.3mb root
```

The `topas` command utilizes the System Performance Measurement Interface (SPMI) from the PTX product. `topas` is similar to the lchmon.c sample code that ships with PTX (in the /var/samples/perfagent/server directory). Since the `topas` command will only give a “snapshot” view of the system resource usage, it has limited usefulness compared to monitoring commands that shows logging output.

The following example will run topas, showing six disks, three network interfaces, two processes and will use five second intervals between snapshot displays:

```
# topas -d6 -n3 -p2 -i5
```

11.1.34 tprof

tprof provides a detailed profile of CPU usage by an application. tprof is like the standard AIX program profilers prof and gprof, in that it provides an estimate of CPU usage by each routine in a program. Unlike prof and gprof, it provides a CPU profile of all processes running during an application, not just the program of interest. Additionally, it provides a profile down to the source statement level, if the associated C or Fortran source files are available. tprof is useful to those whose applications are believed to be CPU-bound and want to know where and why. tprof generates a couple of files, but the one we are interested in is __prof.all.

Shared libraries

In the following example, we examine the usage of shared libraries on the system and get a process report as well. If the LIBPATH environment variable is set for users and elements, then the path might traverse network mounts. tprof can be of help to determine if there are excessive access to libraries that are network mounted or are declared after network mounted paths in the LIBPATH environment variable.

```
# tprof -s&&awk '/Shared Object/,NR==-1{print}' __prof.all|sed '/^$/d'
Shared Object                           Ticks  %   Address  Bytes
 =============                           ===== ==== ======== ======
 /usr/lib/libc.a/shr.o                 18 13.1 d0161720 1c08c5
 /usr/lib/libpthreads.a/shr_xpg5.o           1  0.7 d0004000 20174
   Profile: /usr/lib/libc.a/shr.o
   Total Ticks For All Processes (/usr/lib/libc.a/shr.o) = 18
 Subroutine               Ticks  %   Source   Address Bytes
 =============            ===== ==== ======== ======== ======
 .nl_langinfo                 7  5.1 ../../../../../../../src/bos/usr/ccs/lib/libc/nl_
d0186fa8 78
<<< lines omitted >>>
 .strchr                      1  0.7 strchr.s d0164920 1f4
   Profile: /usr/lib/libpthreads.a/shr_xpg5.o
   Total Ticks For All Processes (/usr/lib/libpthreads.a/shr_xpg5.o) = 1
 Subroutine               Ticks  %   Source   Address Bytes
 =============            ===== ==== ======== ======== ======
 ._init_static_cond           1  0.7 ../../../../../../../src/bos/usr/ccs/lib/l
ibpthreads/cond.c d000d848 1a8
```

Processes

In the following example, we examine processes during one minute (we use the -x option to run the sleep program). Since we are only interested in the top part, we use awk to only show it:

```
# tprof -x sleep 60&&awk '{print}/^           Total/{exit}' __prof.all
Starting Trace now
Starting  sleep 60
Sat Oct  7 17:23:42 2000
System: AIX sp6en0 Node: 4 Machine: 000504936700

Trace is done now
61.381 secs in measured interval
 * Samples from __trc_rpt2
 * Reached second section of __trc_rpt2
Process      PID      TID    Total   Kernel     User   Shared    Other
  =======      ===      ===    =====   ======     ====   ======    =====
      ksh    33814     9125     5687       24     4698      965        0
    hatsd    14192    15741       29       28        1        0        0
      gil     1032     1807       25       25        0        0        0
  hardmon     5414    21475       25       23        2        0        0
<<< lines omitted >>>
  swapper        0        3        1        1        0        0        0
       sh    40550    10859        1        1        0        0        0
       sh    40552    10861        1        0        0        1        0
     cron     9930    19379        1        1        0        0        0
    trace    36646    26483        1        1        0        0        0
    tprof    31584    56453        1        0        0        1        0
    sleep    31584    56453        1        0        0        1        0
  =======      ===      ===    =====   ======     ====   ======    =====
    Total                       6139      398     4744      997        0
```

11.1.35 traceroute

The `traceroute` command attempts to trace the route that an IP packet follows to an Internet host by launching UDP probe packets with a small maximum time-to-live (Max_ttl variable), then listening for an ICMP TIME_EXCEEDED response from gateways along the way. Probes are started with a Max_ttl value of one hop, which is increased one hop at a time until an ICMP PORT_UNREACHABLE message is returned. The ICMP PORT_UNREACHABLE message indicates either that the host has been located, or the command has reached the maximum number of hops allowed for the trace.

The `traceroute` command sends three probes at each Max_ttl setting to record the following:

- Max_ttl value
- Address of the gateway
- Round-trip time of each successful probe

The number of probes sent can be increased by using the -q flag. If the probe answers come from different gateways, the command prints the address of each responding system. If there is no response from a probe within a 3-second time-out interval, an * (asterisk) is printed for that probe.

The `traceroute` command prints an ! (exclamation mark) after the round-trip time if the Max_ttl value is one hop or less. A maximum time-to-live value of one hop or less generally indicates an incompatibility in the way ICMP replies are handled by different network software. The incompatibility can usually be resolved by doubling the last Max_ttl value used and trying again. The following example shows the basic usage of `traceroute` (It is the same example as for `ping`; refer to Section 11.1.24, “ping” on page 302 for more information):

```
# traceroute pollux.ibm.com
trying to get source for pollux.ibm.com
source should be 9.12.0.6
traceroute to pollux.ibm.com (9.46.1.2) from 9.12.0.6 (9.12.0.6), 30 hops
outgoing MTU = 1492
 1  itscpokwf.itso.ibm.com (9.12.0.1)  14 ms  2 ms  2 ms
 2  9.32.41.41 (9.32.41.41)  87 ms  110 ms  112 ms
 3  9.32.44.3 (9.32.44.3)  111 ms  69 ms  64 ms
 4  9.32.1.94 (9.32.1.94)  53 ms  78 ms  88 ms
 5  pollux.cbe.ibm.com (9.46.1.2)  84 ms  99 ms  94 ms
```

11.1.36 trace, trcrpt

AIX provides a vast number of performance monitoring tools to measure the workload of different system resources, such as CPU, memory, disk I/O or network. Out of all these tools, `trace` is probably the most powerful one to monitor system performance, because it is tightly coupled to the kernel and can provide an incredible amount of information.

11.1.36.1 trace

The `trace` facility is useful for observing system activity, particularly that of a running device driver. The `trace` facility captures a sequential flow of time-stamped system events, providing a fine level of detail on system activity. Events are shown in time sequence and in the context of other events.

The `trace` facility is useful in expanding the `trace` event information to understand which, when, how, and even why the event happened. The operating system is shipped with permanent `trace` event points, as can be found in the /etc/trcfmt file. These events provide general visibility to system execution. You can extend the visibility into applications by inserting additional events and providing formatting rules with low overhead. Because of this, the facility is useful, both as a performance analysis tool and as a problem determination tool.

The trace facility is more flexible than traditional system-monitor services that access and present statistics maintained by the system. With traditional monitor services, data reduction (conversion of system events to statistics) is largely coupled to the system instrumentation. For example, the system can maintain the minimum, maximum, and average elapsed time observed for runs of a task and permit this information to be extracted.

The trace facility does not strongly couple data reduction to instrumentation, but it does provide a stream of system events. It is not required to presuppose what statistics are needed. The statistics or data reduction are to a large degree separated from the instrumentation.

This flexibility is important for diagnosing performance or functional problems. For example, netpmon uses the trace facility to report on network activity, including CPU consumption, data rates and response time. The tprof command uses the trace facility to report the CPU consumption of kernel services, library subroutines, application program modules, and individual lines of source code in the application program.

The following sample shows how to extract a trace hook for the vmm trace group (a trace group is a grouping of trace hooks for a related events).

```
# odmget -q 'attribute like vmm_trcgrp' SWservAt |
> awk '$1~/attribute/ && $3~/vmm/{
> trcgrp=substr($3,2,index($3,"_")-2)
> getline
> getline
> gsub("\"","",$3)
> print $3
> }'
1b
```

We use this to run the trace program as follows (-j includes hooks and -k excludes hooks). The next example runs trace on the trace hook 1b (vmm trace group):

```
# trace -j 1b -a
```

The following script lists all defined trace groups from the ODM SWservAt database:

```
#!/bin/ksh

odmget -q 'attribute like *_trcgrp' SWservAt |
  awk '
    $1~/attribute/{
      trcgrp=substr($3,2,index($3,"_")-2)
      getline
      getline
      gsub("\"","",$3)
      print $3
    }'
```

11.1.36.2 trcrpt

The `trcrpt` command reads a `trace` log file, formats the `trace` entries and writes a report to standard output. The following example uses our `trace` file with the `vmm trace` hook 1b above:

```
# trcrpt -O'exec=y' -O'pid=n' -O'tid=n' -O'svc=y' -O'timestamp=3'
System: AIX night4n37 Node: 4
Machine: 006015614C00
Internet Address: 81282366 129.40.35.102
The system contains 4 cpus, of which 4 were traced.
Buffering: Kernel Heap
This is from a 32-bit kernel.

trace -j 1b -a

ID     ELAPSED_SEC     DELTA_MSEC   APPL    SYSCALL KERNEL  INTERRUPT

001    0.000000000       0.000000                  TRACE ON channel 0
                                                   Mon Oct 16 13:08:43 2000
<<< lines omitted >>>
1B2    3.292995634       0.013550                  VMM pagefault:       V.S=04
0D.5D8B7 working_storage modified diskmap (type 2)
1B0    3.293027557       0.031923                  VMM page assign:     V.S=04
0D.5D8B7 ppage=17619A working_storage modified diskmap (type 2)
1B8    3.293027948       0.000391                  VMM vmapped page:    V.S=04
0D.5D8B7 ppage=17619A working_storage modified diskmap (type 2)
1B2    3.293042289       0.014341                  VMM pagefault:       V.S=00
04.18C0 working_storage process_private modified (type 2)
1B0    3.293062430       0.020141                  VMM page assign:     V.S=00
04.18C0 ppage=164435 working_storage process_private modified (type 2)
1B9    3.293062957       0.000527                  VMM zero filled page: V.S=00
04.18C0 ppage=164435
<<< lines omitted >>>
002    3.294807054       0.073813                  TRACE OFF channel 0000 Mon Oct 16 1
```

11.1.37 vmstat

The `vmstat` command focuses on monitoring the resource usage of CPU, memory and I/O. `vmstat` provides very quick and compact information and is the first tool to use for monitoring these components.

The `vmstat` command reports statistics about kernel threads in the run and wait queue, memory, paging, interrupts, system calls, context switches, and CPU activity. The CPU activity is a percentage breakdown of user mode, system mode, idle time, and waits for disk I/O. Keep in mind that the CPU statistics on SMP systems are the average for all of the processors. With `vmstat`, you can not see a per-processor CPU usage on SMP systems, as can be done with the `sar` command.

At each clock interrupt on each processor (100 times a second), a determination is made as to which of four categories (usr/sys/wio/idle) to place the last 10 ms of time. If the CPU was busy in *usr* mode at the time of the clock interrupt, then usr gets the clock tick added into its category. If the CPU was busy in kernel mode at the time of the clock interrupt, then the *sys* category gets the tick. If the CPU was not busy, then a check is made to see if any I/O to disk is in progress. If any disk I/O is in progress, then the *wio* category is incremented. If no disk I/O is in progress and the CPU is not busy, then the *idle* category gets the tick. However, on a SMP system, an idle CPU is only marked as wio if an outstanding I/O was started on that CPU. For example, a node with four CPUs and one thread doing I/O will report a maximum of 25 percent wio time. A node with 12 CPUs and one thread doing I/O will report a maximum of approximately 8 percent wio time.

The kernel maintains statistics for kernel threads, paging, and interrupt activity, which the `vmstat` command accesses through the use of the knlist subroutine and the /dev/kmem pseudo-device driver. The disk I/O statistics are maintained by device drivers. For disks, the average transfer rate is determined by using the active time and number of transfers information. The percent active time is computed from the amount of time the drive is busy during the report.

The following simple, useful example shows us most of what we need by simply running `vmstat` with 5 second intervals:

```
# vmstat 5
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b     avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0 2146520   130   0   4   3  67  238   0 237  243 222 31  7 30 32
 4 11 2146523   135   0   0   0 7008 25490   0 2868 8989 3711 24  9  0 67
 4  9 2145191   122   0   0   0 6812 50234   0 2746 4100 3362 27  8  0 65
 4  9 2145192   124   0   0   0 7602 48337   0 2766 3696 3606 20  8  0 72
 4 11 2145637   133   0   0   0 7109 46670   0 2777 8785 3479 25  8  0 67
 5 13 2145738   134   0   0   0 8267 66832   0 3600 22070 5037 40 12  0 48
 6 10 2144499   119   0   0   0 8348 83568   0 3435 31469 4695 33 14  0 53
 5 10 2144244   131   0   0   0 7325 53636   0 2943 26248 4871 30 12  0 58
 3 11 2144251   133   0   0   0 7309 44760   0 2987 16852 4465 16 12  0 71
^C
```

Column descriptions

We will explain the columns briefly, but the most important fact about the `vmstat` output is that serious performance problems will usually show up as a multiple of high values. If these are analyzed separately it will probably lead to the wrong conclusion. To examine the more complex picture, a thorough understanding on processes' life cycles and resource usage is needed.

r Number of kernel threads placed in run queue. This field indicates the number of runnable threads. This value should be less than five for non-SMP systems. For SMP systems, this value should be less than:

$5 \times (N_{total} - N_{bind})$

Where N_{total} stands for total number of processors and N_{bind} for the number of processors which have been bound to processes, for example, with the `bindprocessor` command. If this number increases rapidly, you should probably look at the applications.

However, to determine if a run queue is too high, it is also necessary to know about the throughput. If the system is performing outstandingly well, a high run queue level can actually be a sign that there is a need for more CPUs.

b Number of kernel threads placed in wait queue (awaiting resource, awaiting input/output). Threads are also located in the wait queue when they are scheduled for execution but waiting for one of their thread pages to be paged in. This value is usually near zero. But, if the run queue value increases, the wait-queue normally also increases. And if this column increases; the wa column is usually high as well.

avm Active virtual memory is defined as the number of virtual-memory working-segment pages that have actually been touched. It is usually equal to the number of paging-space slots that have been assigned. This number can be larger than the number of real page frames in the machine, because some of the active virtual-memory pages may have been written out to paging space.

fre Size of the free list. If the number of pages on the free list drops below minfree, the VMM will steal pages until the free list has been restored to the maxfree value, If the fre value is substantially above the maxfree value, then it is unlikely that the system is thrashing. Thrashing means that the system is continuously paging in and out, searching and freeing. However, if the system is experiencing thrashing, you can be assured that the fre value will be small.

re Pager input/output list. If a page fault occurs and this page is currently on the free list and has not yet been reassigned, this is considered a reclaim since no new I/O request has to be initiated (the page is still in memory). It also includes pages previously requested by VMM for which I/O has not yet been completed or those pre-fetched by VMM's read-ahead mechanism but hidden from the faulting segment. This is not to be confused with the term repage which refers to a page that has already incurred a page-fault (the page could be currently on the free list, filesystem, or in paging space).

pi Pages paged in from paging space. If a page-in occurs, then there must have been a previous page-out for that page. It is also likely in a memory-constrained environment that each page-in will force a different page to be stolen and paged out.

po Pages paged out to paging space. Whenever a page of working storage is stolen, it is written to paging space. If not referenced again, it will remain on the paging device until the process terminates or disclaims the space. Subsequent references to addresses contained within the faulted-out pages results in page faults, and the pages are paged in individually by the system. When a process terminates normally, any paging space allocated to that process is freed.

fr Pages freed (page replacement). As the VMM page-replacement routine scans the Page Frame Table (PFT), it uses criteria to select which pages are to be stolen to replenish the free list of available memory frames. The criteria include both kinds of pages, working (computational) and file (persistent) pages. Just because a page has been freed, it does not mean that any I/O has taken place. For example, if a persistent storage (file) page has not been modified, it will not be written back to the disk. If I/O is not necessary, minimal system resources are required to free a page.

sr Pages scanned by page-replacement algorithm. The VMM page-replacement code scans the PFT and steals pages until the number of frames on the free list is at least the minfree value. The page-replacement code may have to scan many entries in the PFT before it can steal enough to satisfy the free list requirements. With stable, unfragmented memory, the scan rate and free rate may be nearly equal. On systems with multiple processes using many different pages, the pages are more volatile and disjointed. In this scenario, the scan rate may greatly exceed the free rate.

cy Clock cycles by page-replacement algorithm. Since the free list can be replenished without a complete scan of the PFT and because all of the `vmstat` fields are reported as integers, this field is usually zero. If not, it indicates a complete scan of the PFT, and the stealer has to scan the PFT again, because fre is still over the minfree value.

in Device interrupts. To see a detailed list, use the -i option.

sy System calls. Resources are available to user processes through well-defined system calls. These calls instruct the kernel to perform operations for the calling process and exchange data between the kernel and the process. Since workloads and applications vary widely, and different calls perform different functions, it is impossible to say how many system calls per-second are too many. But typically, when the `sy` column raises over a couple of hundred calls per second and per CPU, there should be some further investigations. For this column, it is advisable to have a baseline measurement that gives a count for a normal sy value.

cs Kernel thread context switches. When a thread is given control of the CPU, the context or working environment of the previous thread is saved and the context of the current thread must be loaded (see Section 6.2.1.7, “Process context switching” on page 136 for more information). AIX has a very efficient context switching procedure, so each switch is inexpensive in terms of resources. Any significant increase in context switches should be cause for further investigation, because it might be an indication that thrashing is occurring, especially if the memory indicators are higher (the usage) and lower (the available) than normal.

us User time. When in user mode, a process executes within its application code and does not require kernel resources to perform computations, manage memory or set variables.

sy System time. If a process needs kernel resources, it must execute a system call and is thereby switched to system mode to make that resource available. Generally, if us + sy time is below 90 percent, a single-user system is not considered CPU constrained. However, if us + sy time on a multiuser system exceeds 80 percent, the processes may spend time waiting in the run queue. Response time and throughput might suffer.

id CPU idle time. If there are no processes available for execution (the run queue is empty), the system dispatches a process called wait. This process has priority 127 so if any other process is runnable, the wait process will be preempted until there are no more runnable processes.

wa Basically CPU idle time during which the system had outstanding disk/NFS I/O request(s). A wa value over 25 percent could indicate that the disk subsystem may not be balanced properly, or it may be the result of a disk-intensive workload.

Examples of usage

The following sample shows which device drivers that has generated (received) interrupts:

```
# vmstat -i
priority level    type   count module(handler)
    0       0   hardware     0 i_misc_pwr(fad5c)
    0      24   hardware     0 /etc/drivers/hscsidd(10669ec)
    0      24   hardware     0 /etc/drivers/sdpin(10800d0)
    0      24   hardware     0 i_epow(e4b20)
    0      48   hardware     0 i_scu(fad50)
    1       2   hardware 8202101 /etc/drivers/rsdd(114c8cc)
    3       4   hardware 87983 /etc/drivers/sdpin(1080010)
    3       5   hardware   934 /etc/drivers/middd_loadpin(1181be8)
    3       9   hardware     7 /etc/drivers/entdd(1197dc0)
    3      10   hardware 3622996 /etc/drivers/tokdd(11ac290)
    3      12   hardware 6274350 /etc/drivers/entdd(1197dc0)
    3      14   hardware 469089 /etc/drivers/hscsidd(10669e0)
    4       1   hardware  8394 /etc/drivers/mousedd(1151948)
    4       1   hardware  8952 /etc/drivers/ktsdd(112a540)
    5      62   hardware 158344571 clock(e497c)
   10      63   hardware 749785 i_softoff(e4934)
```

As can be seen in the above example, network and disk has the highest number of interrupts (excluding the clock). The next example uses the `vmstat` summary function two times with a delay in between. It then fixes the output from the two collections and calculates the difference.

```
# (vmstat -s|awk '{print $1}' >/tmp/1;sleep 60;vmstat -s >/tmp/2)
# paste  /tmp/[12]|
> sed 's/\([0-9]*\)/\1:/;
> s/\([0-9]\) \([a-z]\)/\1:\2/g'|
> awk -F: '{printf "%12d %s\n",$2-$1,$3}'
         233 total address trans. faults
           0 page ins
          39 page outs
           0 paging space page ins
           0 paging space page outs
           0 total reclaims
         130 zero filled pages faults
           0 executable filled pages faults
           0 pages examined by clock
           0 revolutions of the clock hand
           0 pages freed by the clock
           0 backtracks
           0 lock misses
           0 free frame waits
           0 extend XPT waits
           0 pending I/O waits
          37 start I/Os
          37 iodones
        4703 cpu context switches
       25521 device interrupts
           0 software interrupts
           0 traps
       10575 syscalls
```

11.1.38 vmtune

The `vmtune` command is mainly used to change operational parameters of the Virtual Memory Manager (VMM) for memory management tuning. `vmtune` can be found in the bos.adt.samples fileset.

It can be used for low level monitoring as well. When running the `vmtune` command without parameters, it displays the current settings for memory management tunables and some statistics as well, as can be seen in the following output:

```
# /usr/samples/kernel/vmtune
vmtune:  current values:
  -p       -P        -r          -R         -f       -F       -N        -W
minperm  maxperm  minpgahead maxpgahead  minfree  maxfree  pd_npages maxrandwrt
 209505   838020       2          8        120      128     524288        0

  -M      -w      -k      -c        -b         -B           -u        -l    -d
maxpin npswarn npskill numclust numfsbufs hd_pbuf_cnt lvm_bufcnt lrubucket defps
838841   73984   18496       1      93        160          9      131072     1

        -s              -n         -S           -h
sync_release_ilock  nokilluid  v_pinshm  strict_maxperm
        0               0          0            0

number of valid memory pages = 1048551  maxperm=79.9% of real memory
maximum pinable=80.0% of real memory    minperm=20.0% of real memory
number of file memory pages = 13254     numperm=1.3% of real memory
```

The most interesting value to monitor is the numperm statistics at the bottom. See Section 6.1.2.6, “minperm and maxperm” on page 123 for more information. Another way of obtaining information with `vmtune` is the -a option, which will give some indication for tuning some of the buffer count values.

```
# /usr/samples/kernel/vmtune -a
       hd_pendqblked = 22
        psbufwaitcnt = 0
        fsbufwaitcnt = 0
```

11.1.39 xmperf, 3dmon

The `xmperf` command can be used to monitor multiple monitoring attributes in one or more screens. Monitoring attributes can easily be added and removed to suit the specific monitoring environment. Performance monitoring is set up by using a hierarchal relationship between the performance monitoring objects. A console is a frame that encompasses performance monitoring instruments. Use the default consoles as templates when creating new consoles.

An instrument is a frame that displays performance data; it defines the type of display used to present the performance data. A value is a performance statistic that is displayed by an instrument. A performance instrument can display multiple values. Depending on the type of instrument, the values are displayed and stacked differently.

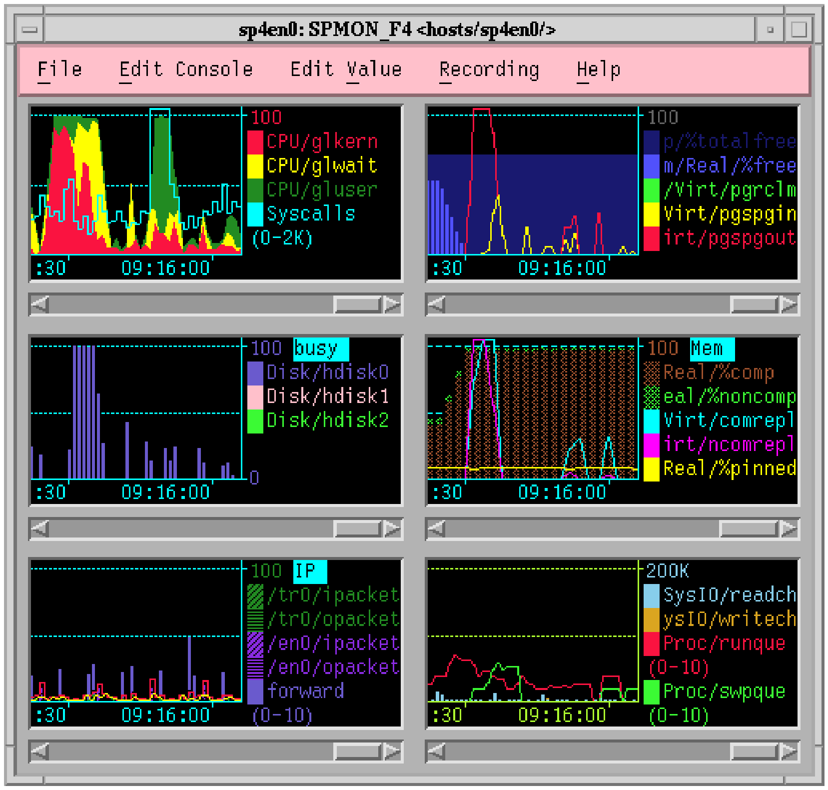

Figure 30. Monitoring a system using xmperf

Figure 30 show a console that encapsulates CPU, VMM, disk activity, memory, network and I/O kernel calls to be monitored.

Figure 31 on page 333 shows a couple of other examples of monitoring screens with `xmperf` and `3dmon`.

Figure 31. xmperf

Part 4. Tuning selected products

Chapter 12. Tivoli Storage Manager (ADSM)

The following chapter seeks to explain the various tunables that can be set to optimize the performance of Tivoli Storage Manager.

Tivoli Storage Manager (TSM) is a client/server program that provides storage management to customers in a multivendor computer environment. TSM provides an automated, centrally scheduled, policy-managed backup, archive and space management facility for file-servers and workstations.

The parameter values discussed in this chapter were tested in an SP environment with 200 Mhz POWER3 nodes with AIX 4.3.3, PSSP 3.2 and TSM 4.1.0 and 4 GB of real memory with internode communication via the SP Switch.

TSM is the follow-on product to IBM's ADSTAR Distributed Storage Manager (ADSM). It is part of Tivoli Storage Management Solutions. More general information about Tivoli storage products can be found at:

`http://www.tivoli.com/products/solutions/storage`

12.1 TSM tunables - quick reference

Table 35 and Table 36 on page 338 provide a quick reference to server and client tuning parameters. All parameters listed in the tables will be discussed in depth later in the chapter.

Table 35. TSM server values in dsmserv.opt

Parameter	Default value	Recommended value
TCPWINDOWSIZE	0	128
TCPNODELAY	no	yes
MAXSESSION	25	40
USELARGEBUFFERS	yes	yes
BUFPOOLSIZE	2048	131072*
LOGPOOLSIZE	512	512
MOVEBATCHSIZE	40	40
MOVESIZETHRESH	500	500
TXNGROUPMAX	40	246

*Refer to Section 12.4.2, “Server options to improve memory performance” on page 342 for further details

Table 36. TSM client values in dsm.sys

Parameter	Default value	Recommended value
TCPWINDOWSIZE	0	128
TCPBUFFSIZE	4	32
TCPNODELAY	no	yes
COMPRESSION	no	no
MEMORYEFFICIENT	no	no
TXBYTELIMIT	2048	25600
RESOURCEUTILIZATION	2	4

12.2 Performance considerations

TSM performance can certainly be influenced by its own tuning parameters, but is also affect by other “external” factors. TSM usually functions in a client server realm supporting many operating systems and working across several networks accepting different communication protocols. All of the factors below can affect TSM performance significantly.

- Client type
- Client speed
- Client activity
- Communication protocol type
- Communication protocol tuning
- Communication controller hardware
- Network activity
- Network speed
- Network reliability
- Server type
- Server speed
- Server activity
- Size and number of files

• Final output repository type (disk, tape, or optical)

If you have an environment with several large application servers which are transferring large quantities of data over a network, there could be benefits from making these machines TSM servers themselves and therefore have local storage cutting out any network bottleneck. Having a group of TSM servers can significantly improve backup/restore times.

12.3 SP network tunables

Table 37 shows the recommended `no` parameters SP server or client running TSM.

Table 37. no parameter values

Parameter	Value
thewall	1310720
sb_max	1310720
rfc1323	1
tcp_mssdflt	32768

Note

thewall can only to set up to half of real memory on AIX 4.3.3.

12.4 Server performance tuning

The default path for the server options file where these are set is:

/usr/tivoli/tsm/server/bin/dsmserv.opt

The following sections discuss how TSM server performance can be improved for the following:

• Network
• Memory
• CPU
• I/O
• TSM Server Database

12.4.1 Server Options to improve network performance

TCPWINDOWSIZE

Purpose: This option specifies the size of the TCP sliding windows in kilobytes. This parameter can also be used in the client options file dsm.sys, which will be discussed later in this chapter.

A larger window size can improve communication performance; this is especially true of fast networks with high latency like the SP switch. However, a window size larger than the buffer space on the network adapter might degrade throughput due to resending packets that were lost on the adapter.

Value: 1 -2048

Default: 0 (This uses the default AIX window size.)

Recommended: 128

TCPNODELAY

Purpose: When set to “no,” this means the server does not allow packets less that the MTU size to be sent immediately over the network. Setting this to “yes” does allow such packets to be sent immediately. For high speed networks, such as the SP switch, setting this to “yes” should improve network performance.

Value: no/yes

Default: no

Recommended: yes (only if you have a high speed network)

MAXSESSION

Purpose: Specifies the maximum number of simultaneous client sessions that can connect with the server. Server performance can be improved by minimizing the number of these sessions, as there will be less network load. The maximum value this option can be is dependent on the available virtual memory and communication resources.

Minimum Value: 2

Default: 25

Recommended: 40

USELARGEBUFFERS

Purpose:	This option increases communication and device I/O buffers. The communication I/O buffer is used during a data transfer with a client session, for example, a backup session, and the disk I/O buffer is used when data is read from a disk storage pool.
Value:	yes/no
Default	yes
Recommended:	yes

12.4.1.1 Scheduling options:

The following settings are not in the dsmserv.opt file and you can query their current value by the `query status` command and change each one using `set`.

MAXSCHEDSESSIONS - Specifies the number of sessions that the server can use for processing scheduled operations. This value is a percentage of the total number of server sessions available via the MAXSESSION parameter. Do not set this to 100 which is 100% of all client sessions; otherwise, you could have a situation where all sessions are being used for running client scheduled backups and you have no available sessions to run a restore.

MAXCMDRETRIES - Maximum number of times that a scheduler on a client node can retry a failed, scheduled command.

RETRYPERIOD - The number of minutes the scheduler on client node waits between retry attempts after a scheduled command fails to process, or there is a failed attempt to contact the server.

QUERYSCHEDPERIOD - Regulates how often client nodes contact the server to check for scheduled work when the server is running in the client polling scheduling mode.

SCHEDMODES - This option specifies if you want *client-polling mode* for the clients to contact the server to pick up scheduled work or whether *server-prompted mode* is used, which means the server contacts the node when it is time to start a scheduled operation. Server-prompted mode means there is less network traffic.

12.4.2 Server options to improve memory performance

BUFPOOLSIZE

Purpose: This is the database buffer pool size; the size is in kilobytes, which provides cache storage. A larger pool means faster access times and better performance, as more of the frequently used database pages can remain cached in memory. However, a larger pool does require more virtual memory. The recommended cache hit percentage is 99 percent. To check the cache hit percentage, use the `query db f=d` command.

Value: 256

Default: 2048

Table 38 gives some recommended sizes for the buffer pool.

Table 38. BufPoolSize recommended sizes

Memory size in MB	BufPoolSize setting
64	32678
128	32678
256	66536
512	131072
1024	131072
2048	131072
4096	131072

SELFTUNEBUFPOOLSIZE

Purpose: If this parameter is set to yes, the TSM server automatically tunes the database buffer pool size. When you activate this option, the server will check the cache hit ratio each time inventory expiration is run. If the cache hit statistics are less than 98%, TSM increases the database buffer pool size. This is not as efficient as setting the buffer pool size manually.

Value: yes/no

Default: no

Recommended: no

LOGPOOLSIZE

Purpose: This option specifies the size of the recovery log buffer pool size in kilobytes. A large recovery log buffer pool may increase the rate at which recovery log transactions are committed to the database, but this will also require more virtual memory. The maximum value of this parameter is determined by the size of the virtual memory.

Minimum Value: 128

Default: 512

Recommended: 512

12.4.3 Server options to improve CPU performance

EXPINTERVAL

Purpose: The TSM server runs automatic inventory expiration, and this option specifies the interval in hours for this process. Inventory expiration removes client backup and archive file copies from the server. Expiration processing is very CPU intensive. Care should be taken to run this when other TSM processes are not active. You need to set EXPINTERVAL to zero to reschedule inventory expiration from the default or run it manually. You can schedule this to run at a quiet time every day via the `schedule expire inventory` command or you can run “expire inventory” manually.

Value: 0 - 336

Default: 24

Recommended: 0

MAXSESSION

Purpose: The number of simultaneous client sessions that can connect to the server can affect the CPU utilization especially, for example, with a gigabit adapter, which can create a lot of work for the CPU.

Minimum Value: 2

Default: 25

Recommended: 40

RESOURCEUTILIZATION

Purpose: This is a parameter that can affect server CPU performance, even though it is set in the client dsm.sys file. This option specifies the number of concurrent sessions that a client can have open with a server. It is only used for backups, not restores.

For example, if set to a value of 4, and there are some larger files to be backed up, it can take some time. The backup of larger files can be moved to separate sessions while the smaller ones can be backed up via one. The extra sessions are automatically opened with the server as needed. If the CPU does end up processing a lot of these concurrent sessions, the backup should be quicker, but only if the CPU can cope with the extra work.

Value: 1 - 10

Default: 2

Recommended: 4

12.4.4 Server options to improve I/O performance

MOVEBATCHSIZE

Purpose: The number of files that should be batched in the same server transaction for server storage pool backup/restore, migration, reclamation or move data operations.

Value: 1 - 256

Default: 40

Recommended: 40 (bigger if there are a lot of small files)

MOVESIZETHRESH

Purpose: A threshold in megabytes for the amount of data moved as a batch within the same server transaction. This parameter works with the *MOVEBATCHSIZE* in the following way. If the number of files in the batch becomes equivalent to the MOVEBATCHSIZE before the cumulative size of the files becomes greater than the MOVESIZETHRESH, then the MOVEBATCHSIZE is used to determine the number of files moved or copied in a transaction. If the cumulative size of the files being gathered for a move or copy operation exceeds the MOVESIZETHRESH value before the number of files becomes equivalent to the MOVEBATCHSIZE, then the MOVESIZETHRESH value is used instead.

Value: 1 - 500

Default: 500

Recommended: 500

TXNGROUPMAX

Purpose: Related to MOVESIZETHRESH and MOVEBATCHSIZE. This gives a hard maximum limit to the number of files that are transferred as a group between the client and server.

Value: 4 - 256

Default: 40

Recommended: 256

12.4.5 Server options to improve TSM server database performance

The performance of a TSM server database can be improved by considering the following:

- It is recommended that the database does not exceed 70 GB in size.
- Run the `unload/load` database commands.

In environments where a large number of small files are being backed up, fragmentation of the database can occur. Running these commands will repack the db pages, reduce the amount of db space used, and also improve database scan performance.

12.5 Client performance tuning

The default path for the client options file where these are set is:

/usr/tivoli/tsm/client/ba/bin/dsm.sys

The main factors that can affect client performance are:

- Network
- Memory
- CPU
- I/O
- Small Filesize

12.5.1 Client options to improve network performance

TCPWINDOWSIZE

Purpose: As for the server, this option specifies the size of the TCPIP sliding window in kilobytes.

Value: 1 - 2048

Default: 0 (this uses the default AIX window size)

Recommended: 128

TCPBUFFSIZE

Purpose: This parameter specifies the size of the internal TCP communication buffer in kilobytes that is used to transfer data between the client node and the server. A large buffer can improve communication performance but does require more memory. A size of 32 KB forces TSM to copy data to its communication buffer and flush the buffer when it fills.

Value: 0 - 32

Default: 4

Recommended: 32

TCPNODELAY

Purpose: As for the server, when set to “no”, this means the server does not allow packets less that the MTU size to be sent immediately over the network. Setting this to yes does allow such packets to be sent immediately. For high speed networks, such as the SP switch, setting this to “yes” should improve network performance.

Value: no/yes

Default: no

Recommended: yes (only if you have a high speed network)

12.5.2 Client options to improve memory performance

COMPRESSION

Purpose: If specified in the client option file as “yes”, it will do the compression in memory. Therefore, you do need larger memory for this. This option compresses files before they get sent to the TSM server. Data Compression will save storage space, network capacity and server cycles, but may have an adverse effect on throughput, which can be significantly slower, than if it is disabled. This is dependent on processor speed. Compression may be beneficial for a fast processor on a slow network.

Value: yes/no

Default: no

Recommended: no

MEMORYEFFICIENT

Purpose: When doing a backup, an object list will have to be created. The more files and directories, the larger the list. This requires a lot of memory; specifying this option as "yes" will create a more efficient memory allocation, although it can slow down performance significantly, so it should only be used when really needed.

Value: yes/no

Default: no

Recommended: no

TXNBYTELIMIT

Purpose: Specifies the number of kilobytes the client can buffer together in a transaction before it sends data to the server. Increasing the amount of data per transaction will increase the recovery log requirements on the server. Increasing the amount of data per transaction will also result in more data having to be retransmitted if a retry occurs, which can negatively affect performance.

Value: 300 - 25600

Default: 2048

Recommended: 25600

12.5.3 Client options to improve CPU performance

RESOURCEUTILIZATION

Purpose: The smaller the value, the better for CPU performance. The larger the value, the larger the number of concurrent sessions allowed with the server, and the more stress is placed on CPU resources.

Value: 1 - 10

Default: 2

Recommended: 4

COMPRESSION - Setting this to yes can adversely affect CPU performance.

> **Note**
>
> When doing a backup, building an object list will use CPU resources. Therefore, the larger the list, the more CPU resources will be affected.

12.5.4 Client options to improve I/O performance

RESOURCEUTILIZATION - Unlike for CPU performance, setting a larger value for RESOURCEUTILIZATION improves I/O performance, for example, if you have multiple tape drives, it can help by doing parallel transfers. It is not used for restores; multiple logical nodes still may be a good way to improve I/O performance for those operations.

12.5.5 Client options to improve small file size performance

Both the parameters RESOURCEUTILIZATION and TXNBYTELIMIT set to a higher value will increase performance on a system which has to deal with a large number of small files.

12.6 Some test results

We first set up a local SCSI disk pool on the TSM server with another SP node (exnode1) set up as a client. Communication between server and client was over the SP switch.

Initial testing had, as the default, dsmserv.opt and dsm.sys files with no parameters changes (except *Servername* and *TCPServeraddress* in dsm.sys), as shown in Figure 32 on page 349.

```
dsm.sys file:
   SErvername TSMTEST
   COMMmethod TCPip
   TCPPort   1500
   TCPServeraddress 192.167.4.34

dsmserv.opt:
   COMMmethod TCPIP
   COMMmethod SHAREDMEM
   COMMmethod HTTP
```

Figure 32. Default client and server files

On backing up, the following results were observed:

```
Total number of objects inspected:   17,182
Total number of objects backed up:   17,159
Total number of objects updated:          0
Total number of objects rebound:          0
Total number of objects deleted:          0
Total number of objects expired:          0
Total number of objects failed:           0
Total number of bytes transferred:   515.00 MB
Data transfer time:                   31.61 sec
Network data transfer rate:       16,683.38 KB/sec
Aggregate data transfer rate:      2,680.59 KB/sec
Objects compressed by:                   0%
Elapsed processing time:           00:03:16
```

Restoring gave the following output:

```
Total number of objects restored:    15,302
Total number of objects failed:           0
Total number of bytes transferred:   471.10 MB
Data transfer time:                2,948.85 sec
Network data transfer rate:          163.59 KB/sec
Aggregate data transfer rate:        145.22 KB/sec
Elapsed processing time:           00:55:21
```

This meant that it had taken 55 minutes to only back up half a gigabyte to a disk storage pool.

We then changed the client dsm.sys file, but left the server dsmserv.opt file with its default values, as shown in Figure 33 on page 350:

```
dsm.sys
   SErvername TSMTEST
   COMMmethod TCPip
   TCPPort   1500
   TCPServeraddress 192.167.4.34
   TCPWindowSize 128
   TCPBuffSize 32
   TCPNoDelay yes
   TXNByteLimit 25600
```

Figure 33. Changed client dsm.sys file

The backup ran quicker:

```
Total number of objects inspected:   17,184
Total number of objects backed up:   17,161
Total number of objects updated:          0
Total number of objects rebound:          0
Total number of objects deleted:          0
Total number of objects expired:          0
Total number of objects failed:           0
Total number of bytes transferred:   515.01 MB
Data transfer time:                   43.33 sec
Network data transfer rate:        12,168.53 KB/sec
Aggregate data transfer rate:       5,134.88 KB/sec
Objects compressed by:                    0%
Elapsed processing time:           00:01:42
```

However, no change was seen in the speed of the restore:

```
Total number of objects restored:    15,302
Total number of objects failed:           0
Total number of bytes transferred:   471.10 MB
Data transfer time:                2,956.55 sec
Network data transfer rate:          163.16 KB/sec
Aggregate data transfer rate:        145.00 KB/sec
Elapsed processing time:           00:55:26
```

We then made changes to the server dsmserv.opt file, as shown inFigure 34 on page 351; all other parameters not listed were at default value.

```
dsmserv.opt file:
   COMMmethod TCPIP
   COMMmethod SHAREDMEM
   COMMmethod HTTP
   TCPWindowSize 128
   BUFFPoolSize 131072
   TXNGroup 256
```

Figure 34. Changed server dsmserv.opt file

Once we had made the changes to both the server and client files, we saw a dramatic increase in the restore rate: The *elapsed processing time* went from 55 minutes 26 seconds to 6 minutes 21 seconds. For example:

```
Total number of objects restored:    15,302
Total number of objects failed:           0
Total number of bytes transferred:   471.10 MB
Data transfer time:                    6.82 sec
Network data transfer rate:        70,717.83 KB/sec
Aggregate data transfer rate:      1,265.62 KB/sec
Elapsed processing time:           00:06:21
```

We then added the line:

RESOURCEUTILIZATION 4

to the client dsm.sys file. The backup did run quicker. For example:

```
Total number of objects inspected:   17,187
Total number of objects backed up:   17,164
Total number of objects updated:          0
Total number of objects rebound:          0
Total number of objects deleted:          0
Total number of objects expired:          0
Total number of objects failed:           0
Total number of bytes transferred:   530.45 MB
Data transfer time:                   72.08 sec
Network data transfer rate:        7,535.87 KB/sec
Aggregate data transfer rate:      6,483.77 KB/sec
Objects compressed by:                    0%
Elapsed processing time:           00:01:23
```

To see how many sessions the client was actually using, we ran the `q session` from the server:

```
tsm: SERVER1>q se
Session established with server SERVER1: AIX-RS/6000
  Server Version 4, Release 1, Level 0.0
  Server date/time: 11/10/00   06:39:12  Last access: 11/10/00   05:30:21


  Sess Comm.  Sess     Wait   Bytes   Bytes Sess  Platform Client Name
Number Method State    Time    Sent   Recvd Type
------ ------ ------ ------ ------- ------- ----- -------- -----------------
     9 Tcp/Ip IdleW   56 S      968     644 Node  AIX      EXNODE1
    10 Tcp/Ip RecvW    0 S      657  52.3 M Node  AIX      EXNODE1
    11 Tcp/Ip Run      0 S      120     113 Admin AIX      ADMIN
```

As a final test, we decided to run a restore with the changes in the server dsmserv.opt file but with a default client dsm.sys file. Once again, the restore was significantly slower, but not as slow as having a changed client dsm.sys file and a default server dsmserv.opt file. For example:

```
Total number of objects inspected:   37,489
Total number of objects backed up:   16,774
Total number of objects updated:          0
Total number of objects rebound:          0
Total number of objects deleted:          0
Total number of objects expired:          0
Total number of objects failed:           0
Total number of bytes transferred:   538.16 MB
Data transfer time:                1,966.39 sec
Network data transfer rate:          280.24 KB/sec
Aggregate data transfer rate:        266.26 KB/sec
Objects compressed by:                    0%
Elapsed processing time:           00:34:29
```

It was our observation that significant improvements in performance, especially with the time taken for a restore, could be achieved by tuning TSM server and client files.

Chapter 13. VSD

This chapter discusses the performance and tuning for IBM Virtual Shared Disk (VSD). It describes VSD parameters and the two new enhancements in VSD 3.2, concurrent VSD and VSD usage of KLAPI, which can help improve the performance of your VSD subsystem.

Though an overview of VSD is provided, it is assumed that the reader has some basic knowledge and experiences in installation, configuration and managing of VSD.

For more detailed VSD information, consult *PSSP: Managing Shared Disks*, SA22-7279.

13.1 Overview

VSD is a subsystem that allows data in logical volumes on disks physically connected to one node to be transparently accessed by other nodes. VSD only supports raw logical volumes. GPFS, described in Chapter 15, “Web applications” on page 393, provides a similar function, global access to data, but for the file system.

Figure 35 shows a sample configuration of a simplified VSD implementation.

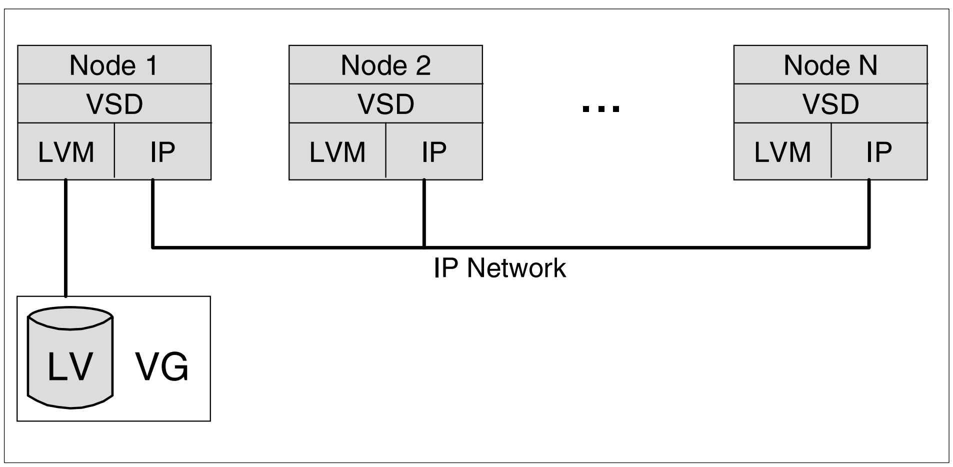

Figure 35. VSD implementation

Here node 1 is called a *VSD server* node, because it provides read and/or write access to data on the disks it owns. Node 2 to Node n are called *VSD client* nodes, because they request access to the data on VSD server nodes.

The client node can access VSD on the server node via the VSD layer, which checks whether this node owns the VSD. If it is, the VSD layer routes the request to the LVM layer to access the disk. If not, it routes the I/O requests to the server that owns that VSD. The VSD server node accesses the data and sends the results back to the client.

13.2 Tuning

The main VSD tunable areas are:

- Application buffer allocation
- I/O subsystem
- Switch adapter
- VSD cache buffer
- VSD request blocks
- VSD pbufs
- Buddy buffer
- Maximum I/O request size

The actual processing capability required for VSD service is a function of the application I/O access patterns, the node model, the type of disk, and the disk connection.

If you plan to run time-critical applications on a VSD server, remember that servicing disk requests from other nodes might conflict with the demands of these applications.

Make sure that you have sufficient resources to run the VSD program efficiently. This includes enough buddy buffers of sufficient size to match your file system block size, as well as sufficient rpoolsize and spoolsize blocks in the communications subsystem.

13.2.1 Application considerations

Your application should put all allocated buffers on a page boundary.

If your I/O buffer is not aligned on a page boundary, the VSD device driver will not parallel I/O requests to underlying VSDs, and performance will be degraded.

In addition, the KLAPI protocol, which is mentioned later in the chapter, requires the data to be page aligned in order to avoid a data copy.

13.2.2 I/O subsystem considerations

The VSD device driver passes all its requests to the underlying LVM subsystem. Thus, before you tune VSD, verify that the I/O subsystem is not the bottleneck. See Section 6.3, “I/O” on page 141 for more information on how to tune the I/O subsystem.

If an overloaded I/O subsystem is degrading your system’s performance, tuning the VSD will not help. Instead, consider spreading the I/O load across more disks or nodes.

With AIX 4.3.3, you may consider using the striping and mirroring logical volume.

13.2.3 SP switch considerations

Several considerations need to be made regarding the SP switch:

mbufs and thewall
mbufs are used for data transfer between the client and the server nodes by VSD subsystem's own UDP-like Internet protocol. thewall controls the maximum amount of memory to be used for communication buffer area.

You need not do anything with thewall because the default setting for AIX 4.3.3 is sufficient for VSD usage.

Switch Send/Receive Pool
If you are using the switch (css0) as your communications adapter, the IBM VSD component uses mbuf clusters to do I/O directly from the switch's send and receive pools.

If you suspect that the switch pool may not be enough, use the appropriate `vdidl3` command (The exact command to be used depends on the type of switch adapter you are using. This is mentioned in more detail in Section 5.2.6, “Sizing send and receive pool requirements” on page 107; you can also use `errpt` to check the error log.

If you see the line:

```
IFIOCTL_MGET(): send pool shortage
```

in the error log, you should consider increasing the size of the send and receive pools.

Refer to Section 5.2, “SP Switch and SP Switch2 adapter tuning” on page 99 on how to display and change send and receive pool size.

Be aware that for the SP switch, you need to reboot the node after changing the size, but for the SP switch2, you do not need to reboot.

maximum IP message size

This defines the largest size of the packets that the VSD software sends between the client and the server.

Its value can vary between 512 bytes and 65024 bytes (63.5KB). The default setting is 61440 bytes, which is the recommend value.

Larger values of max_IP_msg_size will result in fewer packets being required to transfer the same amount of data and generally have better performance than using a small packet size.

To display the maximum IP message size, use the following command:

```
# dsh 'statvsd|grep max'
```

It is recommended that you set the maximum IP message size to 61440 when you configure a VSD node and do not change it.

However, in case you need to change it, use the `updatevsdnode` command to change the value in the SDR (so that it persists across nodes reboot) and then use `ctlvsd` to change the current value:

```
# updatevsdnode -n ALL -M 61440
# dsh ctlvsd -M 61440
```

maximum buddy buffer size

Ensure that the maximum buddy buffer size is at least 256 KB.

Refer to “Buddy buffers” on page 359 for more details.

13.2.4 VSD parameters:

Here we discuss various parameters that can affect VSD performance.

VSD cache buffer

Each VSD device driver, that is, each node, has a single cache buffer, shared by cachable VSDs configured on and served by the node.

The cache buffer is used to store the most recently accessed data from the cached VSDs (associated logical volumes) on the server node. The objective is to minimize physical disk I/O activity. If the requested data is found in the cache, it is read from the cache, rather than the corresponding logical volume.

Data in the cache is stored in 4 KB blocks. The content of the cache is a replica of the corresponding data blocks on the physical disks. Write-through cache semantics apply; that is, the write operation is not complete until the data is on the disk.

When you create VSDs with VSD perspectives or the `createvsd` command, you specify the *cache* option or the *nocache* option.

IBM suggests that you specify nocache in most instances (especially in the case of read-only or other than 4 KB applications) for the following reasons:

- Requests that are not exactly 4 KB and not aligned on a 4 KB boundary will bypass the cache buffer, but will incur the overhead of searching the cache blocks for overlapping pages.
- Every 4 KB I/O operation incurs the overhead of copying into or out of the cache buffer, as well as the overhead of moving program data from the processor cache due to the copy.
- There is overhead for maintaining an index on the blocks cached.

If you are running an application that involves heavy writing followed immediately by reading, it might be advantageous to turn the cache buffer on for some VSDs on a particular node.

To display whether a VSD named koavsd1n7 has enabled VSD cache or not, use the command `lsvsd -l` and see the “option” column:

```
root@sp6n07:/: lsvsd -l koavsd1n7
minor  state server lv_major lv_minor vsd-name option size(MB) server_list
 1      ACT    7      38       1      koavsd1n7 nocache 32             7
```

To enable the cache for koavsd1n7, use the command `updatevsdtab` and then use the `ha.vsd refresh` command:

```
root@sp6en0:/: updatevsdtab -v koavsd1n7 -o cache
root@sp6en0:/: dsh ha.vsd refresh
n1: Quorum is already 6
n1: The request for subsystem refresh was asynchronously started.
...
n14: Quorum is already 6
n14: The request for subsystem refresh was asynchronously started.
root@sp6en0:/:
```

Be aware that Concurrent VSD does not support VSD cache.

VSD request blocks

One or more request blocks are allocated for each VSD request on a node regardless whether the request is to the local or remote VSDs. They enable VSD to keep track of the current outstanding VSD requests.

Since a large request may be broken up into smaller subrequests, the number of request blocks may be several times higher than the actual number of pending read/write requests.

VSD_request_blocks or vsd_request_count specifies the maximum number of request blocks that are available on a node. Each request block is approximately 76 bytes.

Prior to VSD 3.2, this was one of the most important tunables for VSD. In VSD 3.2, this parameter is now internally controlled and tuned by the VSD subsystem.

As a consequence, VSD ignores the value set in the SDR or the value on the -r option in the `updatevsdnode` command (if set).

VSD pbufs

VSD pbufs or rw_request_count specifies the maximum number of pending device requests for a specific VSD on its server node.

Each VSD, regardless of its activity and whether the node is a client or a server, is allocated these pbufs. Each pbuf is 128 bytes long.

Prior to VSD 3.2, this was one of the most important tunables for VSD. In VSD 3.2, this parameter is now internally controlled and tuned by VSD subsystem.

As a consequence, VSD ignores the value set in the SDR or the value on the -p option in the `updatevsdnode` command.

Note

Prior to VSD 3.2, VSD runs in interrupt mode, so it must have all resources ready to be used. As a result, request block or pbuf are pre-allocated to the maximum amount specified and pinned in the memory.

This can place quite a heavy load on the kernel heap if the parameters are not specified correctly, for example, if you specify too many pbufs, the system can crash due to the lack of kernel heap. Also, this approach of memory allocation does not make an optimal usage of memory.

With VSD 3.2, VSD now runs as a normal process/thread; this enables it to dynamically acquire memory as needed. As a result, request count and pbuf are now allocated and pinned as needed.

Start from 0; it is allocated on demand (and kept for future use) until a low water mark is reached. After that, the new allocation will be returned to the system after usage in order to minimize the amount of pinned memory.

In case the allocation goes over a high water mark, the request will be queued and the corresponding value of “request queue waiting for a request block or pbuf” will be incremented.

However, this should rarely happen, because these water mark values had been tested with heavy GPFS workload in the lab with no incidents of request queue waiting.

This also helps save memory, instead of having 48 pbufs per each LV; we now have just one shared pbuf pool that is used by all LVs on a node.

Buddy buffers

The VSD server node uses a pinned memory area called buddy buffers to temporarily store data for I/O operations originating at a client node.

In contrast to the data in the cache buffer, the data in a buddy buffer is purged immediately after the I/O operation completes.

Buddy buffers are allocated in powers of two. If an I/O request size is not a power of two, the smallest power of two that is larger than the request is allocated. For example, for a request size of 24 KB, 32 KB is allocated on the server.

Note

Buddy buffers are only used on the servers. On client nodes you may want to set max_buddy_buffers to 1.

Buddy buffers are used only when a shortage in the switch buffer pool occurs or when the size of the request is greater than the maximum IP message size (60 KB).

KLAPI does not use the switch buffer pool; it always uses buddy buffers.

There are three parameters associated with the buddy buffer:

- Minimum buddy buffer size allocated to a single request
- Maximum buddy buffer size allocated to a single request
- Total size of the buddy buffer

If you are using the switch as your adapter for VSD, we recommend setting 4096 (4 KB) and 262144 (256 KB), respectively, for minimum and maximum buddy buffer size allocated to a single request.

To display the minimum buddy buffer size, use the command `vsdatalst -n` and see the “Buddy Buffer minimum size” column.

To set the minimum buddy buffer size to 4 MB (or 4096 KB) on node 1, use the command:

```
# updatevsdnode -n 1 -b 4096
```

To display the maximum buddy buffer size, use the command `vsdatalst -n` and see the “Buddy Buffer maximum size” column.

To set the maximum buddy buffer size to 256 MB (or 262144 KB) on node 1, use the command:

```
# updatevsdnode -n 1 -x 262144
```

The total amount of memory that is available for buddy buffers is specified by the number of maximum sized buddy buffers multiplied by the maximum buddy buffer size.

To display the number of maximum sized buddy buffer, use the command `vsdatalst -n` and see the “Buddy Buffer size: maxbufs” column.

We recommend you set the initial maximum number of buddy buffer in the range of 32 to 96 (where the maximum buddy buffer size is 256 KB).

To set the number of maximum sized buddy buffer to 96 on all nodes, use the following commands:

```
# updatevsdnode -n ALL -s 96
```

Note

To make any buddy buffer related changes effective, you need to:

```
# ...stop the application that is using VSD...
# dsh hc.vsd stop
# dsh ha.vsd stop
# ucfgvsd -a
# ucfgvsd VSD0
# ...updatevsdnode command...
# dsh ha_vsd reset
# ...start the application that is using VSD...
```

If the `statvsd` command consistently shows *requests queued waiting for buddy buffers* and the switch buffer pool is not at the maximum size (16 MB for SP switch, 32 MB for SP switch2), instead of increasing your buddy buffers, consider increasing your switch buffer pool size.

In case the switch buffer pool is already at the maximum size and you are sure that there is no I/O bottleneck, monitor the value of *buddy buffer wait_queue size* from the `statvsd` command.

buddy buffer wait_queue size, a new output from the `statvsd` command, is the current number of VSD requests waiting in the queue. Its value varies according to the demand for buddy buffer at the time the command runs.

If the output consistently shows a value of n for the buddy buffer wait_queue size, you can increase the number of buddy buffer by n.

Maximum I/O request size

The value you assign to maximum buddy_buffer_size and the max_IP_msg_size limits the maximum size of the request that the IBM VSD subsystem sends across the switch.

For example, if you have:

- A request from a client to write 256 KB of data to a remote VSD
- A maximum buddy buffer size of 64 KB
- A maximum IP message size of 60 KB

the following transmission sequence occurs:

1. Since the VSD client knows that the maximum buddy buffer size on the VSD server is 64 KB, it divides the 256 KB of data into four 64 KB requests.
2. Since the maximum size of the packet is 60 KB, each 64 KB block of data becomes one 60 KB packet and one 4 KB packet for transmission to the server via IP.
3. At the server, the eight packets are reassembled into four 64 KB blocks of data, each in a 64 KB buddy buffer
4. The server then has to perform four 64 KB write operations to disk and return four acknowledgements to the client.

A better scenario for the same write operation would use the maximum buddy buffer size of 256 KB:

- The same 256 KB client request to the remote VSD
- The maximum buddy buffer size of 256 KB
- The maximum IP message size of 60 KB

This produces the following transmission sequence:

1. Since the request can fit into a buddy buffer on the VSD server, the client does not need to divide the request into a smaller size.
2. Since the maximum size of the packet is 60 KB, the 256 KB request becomes four 60 KB packets and one 16 KB packet for transmission to the server via IP.
3. At the server, the five packets are reassembled into one 256 KB block of data in a single buddy buffer.
4. The server then performs one 256 KB write operation and returns an acknowledgement to the client.

The second scenario is preferable to the first because the I/O operations at the server are minimized.

This example also shows that you should configure the buddy buffer size to match the request from the application and that the larger buddy buffer size can give better performance.

13.2.5 Summary of VSD tuning recommendations

Here is a summary for tuning the VSD subsystem:

- Make sure that the application buffer allocation is aligned on a page boundary.

- Make sure that the I/O subsystem is not a bottleneck.
- Use the maximum size of switch send and receive pool.
- Set max_IP_msg_size to 61440 to maximize the switch packet size.
- Set the maximum buddy buffer size to at least 256 KB.
- On VSD server, set the number of buddy buffer to 32-96 initially then monitor with `statvsd` command for buddy buffer wait_queue size.
- On VSD client, set the number of buddy buffer to 1.

13.3 Other VSD considerations

In this section, we describe the two new features in VSD 3.2, concurrent VSD and VSD usage of KLAPI.

By exploiting these features, it is very likely that you can improve the performance of your VSD subsystem.

13.3.1 Concurrent VSD

VSD 3.2 includes Concurrent VSD (CVSD), which allows you to use multiple servers (currently limited to two) to satisfy disk requests by taking advantage of the services of Concurrent Logical Volume Manager (CLVM) supplied by AIX.

Note

Since CVSD makes use of CLVM, it has the same limitation as CLVM, that is, no mirror write consistency and bad block relocation.

VSD cache is not implemented for CVSD because the overhead of making sure the cache coherency outweighs its benefits.

Only two servers and SSA disks are supported in the current release.

CVSD has potentials to improve VSD performance because:

- Not only one, but two VSD servers can now serve the requests for VSD. This can help speed up the response time.
- Each VSD client in the system keeps track of which VSD server they used to access CVSD the last time; the next time it needs to access that CVSD again, it will use another server. This helps distribute the I/O requests evenly among the two servers.

CVSD also helps make VSD more highly available.

- Prior to VSD 3.2, VSD will not be available during the takeover of a primary node and during the takeback. With CVSD, this is improved because CVSD is always available via the surviving node.
- When a node fails, Group Services will inform the RVSD subsystem. The surviving node will fence the failed node so that it cannot access the CVSD. The other client nodes will also be informed so that they know it is not possible to get to that CVSD from the failed node anymore.

  When the failed node comes back, it will be unfenced from the CVSD. Other nodes will be informed that it is now possible to get to the CVSD from this node.

  This reduces the takeover and takeback time, because all that is needed to do is to run `fencevg` or `unfencevg` command; there is no need to run `varyoffvg`, `varyonvg` or `importvg` at all.

To implement CVSD:

1. Add the node to a cluster with the -c clustername option of the `updatevsdnode` command.

> **Note**
>
> Only two nodes are allowed in a cluster.
>
> A node can belong to only one cluster.
>
> Use `vsdatalst -c` shows which node is in which cluster.

2. Reboot.

> **Note**
>
> A reboot is needed here since VSD will set the node number in the node's ssa device, and this setting requires reboot to make it effective.

3. To create a CVSD, specify -k CVSD option on the `createvsd` command.

13.3.2 VSD usage of KLAPI

The Kernel Low-level Application Programming Interface (KLAPI) protocol has been introduced to meet the growing performance requirements of IBM's GPFS and VSD.

The overhead of GPFS has been dominated by multiple data copies. Studies have concluded that elimination of a data copy in GPFS is required to reduce the overhead and improve node throughput. With many large RS/6000 SP systems running GPFS with large amounts of data, these performance enhancements are a necessity.

With data being written and read becoming larger, the VSD transport services have been improved through KLAPI.

KLAPI is an efficient transport service for communication between clients and servers and has been developed to aid VSD. KLAPI is a zero copy transport protocol which supports fragmentation and reassemble of large messages, packet flow control and recovery from lost packets.

The Message size can be as large as GPFS block size.

Packet flow control is introduced to prevent switch congestion when many nodes send to one, for example, in the VSD environment. This helps reduce the VSD retries which can significantly impact performance.

In short, KLAPI provides transport services to kernel subsystems that need to communicate via the SP switch. KLAPI semantics for active messages are similar to those for user space LAPI.

To exploit KLAPI, VSD utilizes KLAPI transport services and provides its own buffer management on the server side. For GPFS, VSD manages direct DMA between the buddy buffer and the GPFS pagepool using KLAPI services.

To utilize KLAPI transport services, VSD replaces calls to its own send and receive functions with calls to the KLAPI active message interface.

The following examples are simplified to illustrate the difference in the data flow when a GPFS application node (which is a VSD client) requests a write to VSD server using IP and KLAPI.

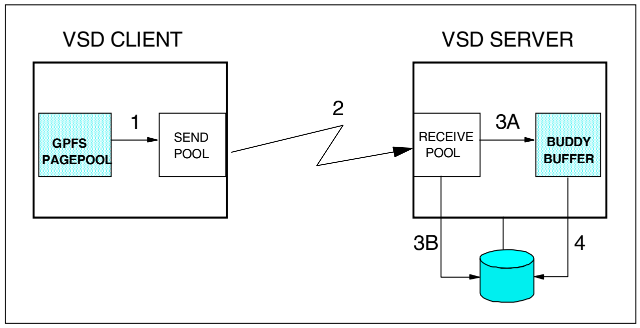

Figure 36. Write data flow: VSD using IP

Using IP, the data from the application is copied to the GPFS pagepool, which is then copied to the send pool (1) (see Figure 36 for an overview). It is then transmitted to the VSD server's receive pool (2).

If the data is less than 60K (max IP message size), it is written to the disk (3B), if not, it is copied to the buddy buffer (3A) and then written to disk (4).

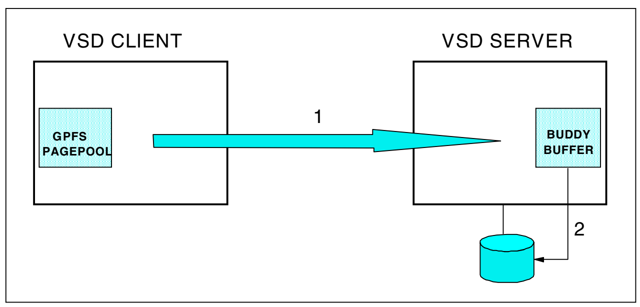

Figure 37. Write data flow: VSD using KLAPI

But using KLAPI, after the data from the application is copied to the GPFS pagepool, VSD transfer the data from the pagepool on the client to the buddy buffer on the server directly via DMA.

From the examples, you can see that with KLAPI, there is zero copy, because the data will be transferred directly between the buddy buffer and the application buffer (GPFS pagepool in this case) See Figure 37 for an overview.

This helps reduce a lot of CPU usage spent in copying data from one buffer to another. This can also help improve the performance, because the application now has more available CPU time.

For VSD, one important difference between IP and KLAPI is that VSD with KLAPI has flow control, while VSD with IP does not.

With IP, a lot of client nodes can send the packet to the server node and find out that the receive pool on the server node is full. Those packets will be dropped and need to be re-transmitted. This uses (and wastes) a lot of switch network bandwidth.

With KLAPI, before a client node can start sending to the server node, a buddy buffer has to be allocated and prepare for DMA access. So when the client sends the packet, it will not be dropped, but put into that buddy buffer.

This also means that if a server runs out of buddy buffers, the client will know about this and it will not send the packet but wait until a buddy buffer is available. This helps reduce the amount of unnecessary traffic (retry) on the switch network and provides flow control for the VSD subsystem.

To enable KLAPI:

1. Use `dsh 'statvsd | grep vsdd'` to verify whether KLAPI is enabled. If the output says "VSD driver (vsdd): KLAPI interface" then KLAPI is enabled.
2. Use `dsh ha.vsd stop` to stop the RVSD subsystem.
3. Use `dsh ctlvsd -l on` to enable VSD to use KLAPI.
4. Use `dsh ha_vsd reset` to restart the RVSD subsystem.

Note

KLAPI is not supported for use by Oracle Parallel Server. This is currently a testing limitation only.

KLAPI protocol can only be used between nodes with VSD 3.2 installed.

KLAPI always uses buddy buffers (even for small requests).

Chapter 14. GPFS

This chapter discusses performance and tuning for IBM General Parallel File System for AIX (GPFS). It provides an overview of GPFS, and describes various parameters in GPFS and its related subsystems that can have performance implications in GPFS.

Though an overview of GPFS is provided, it is assumed that the reader has some basic knowledge and experience in installation, configuration, and managing of GPFS.

For more detailed GPFS information, consult the following manuals:

- *IBM General Parallel File System for AIX: Concepts, Planning, and Installation,* GA22-7453
- *IBM General Parallel File System for AIX: Administration and Programming Reference*, SA22-7452
- *IBM General Parallel File System for AIX: Problem Determination Guide,* GA22-7434
- *IBM General Parallel File System for AIX: Data Management API Guide,* GA22-7435

Also, check the following redbooks:

- *GPFS: A Parallel File System*, SG24-5165
- *GPFS: Sizing and Tuning for GPFS*, SG24-5610

14.1 Overview

GPFS is implemented as a standard AIX Virtual File System, which means that most applications using standard AIX VFS calls (such as JFS calls) will run on GPFS without any modification.

GPFS allows parallel applications simultaneous access to the same files, or different files, from any node in the configuration while maintaining a high level of data availability. It offers an extremely highly available file system by utilizing both hardware and software redundancy where appropriate (for example, disk takeover when a node fails via Recoverable Virtual Shared Disk (RVSD), replicated log files, selectable data/metadata replication at the file system level, and so on).

GPFS achieves a very high level of performance by having multiple nodes acting in cooperation to provide server functions for a file system. This solves the problem of running out of server capacity that occurs with NFS more servers or disks can be added to improve performance.

Having multiple servers will not help much unless we can make sure that all nodes are working together in parallel and the system workload is spread out across them evenly, so none of them become a bottleneck.

The following are some examples of what GPFS does to spread the workload across nodes:

- GPFS stripes data across disks on multiple nodes
- GPFS allows more disks and nodes to be added later
- GPFS can re-stripe the file system
- GPFS does not assign the same node to be the file system manager, if possible

When you create a file system in GPFS, you can specify a *block size* of 16 KB, 64 KB, 256 KB, 512 KB, or 1024 KB. Block size is the largest amount of data that is accessed in a single I/O operation. Each block is divided into 32 *subblocks* which is the smallest unit of disk space that can be allocated. For example, using a block size of 256 KB, GPFS can read as much as 256 KB in a single I/O operation and small files occupy at least 8 KB of disk space.

When GPFS writes data to the file system, it stripes the data into many blocks of the block size, then writes each block to a disk in the file system. You can specify the method GPFS should use for allocating each block to disk, for example round robin or random, when you create the file system.

GPFS achieves the goal of being a highly available file system through its ability to recover from various hardware and software failures. In most cases, the end users can continue operations with only a slight delay or performance degradation.

With the help of RVSD and twin-tailed disk or SSA looping connection, GPFS is able to tolerate various hardware failures such as node failure, switch adapter failure, disk adapter failure, and disk cable failure.

Furthermore, with the implementation of Concurrent Virtual Shared Disk (CVSD), now there can be two active VSD servers for each VSD. Thus, even if there is a failure with one server, that VSD is available via the other surviving server. This greatly enhances VSD availability.

Note

CVSD does not replace RVSD. RVSD is required for CVSD operation. CVSD just helps make the failover time shorter.

GPFS can handle disk failures by:

- Using RAID-5 disks.
- Implementing disk mirroring.
- Using data and/or metadata replication.

  When you create the GPFS file system, you can specify whether you would like to replicate the data and/or the metadata for the file system. You can select up to two replicas for data and metadata.

  GPFS ensures that a copy of replica will always be available even when there is a hardware failure (In certain cases, it may be able to survive multiple failures).

Even when we do not use RAID-5 disk or implement disk mirroring and data/metadata replication (and multiple failures occur), GPFS can still provide access to the file as long as all the required metadata to access the file and its data can still be accessed.

Moreover, the file system integrity in GPFS is always maintained because GPFS replicates the log file of each node to another one. Thus when a node fails, the other node can be used to recover from the failure. This not only allows the file system integrity to be maintained, but also allows the file system operation to continue with no disruption.

An example of a GPFS configuration is provided in Figure 38 on page 372.

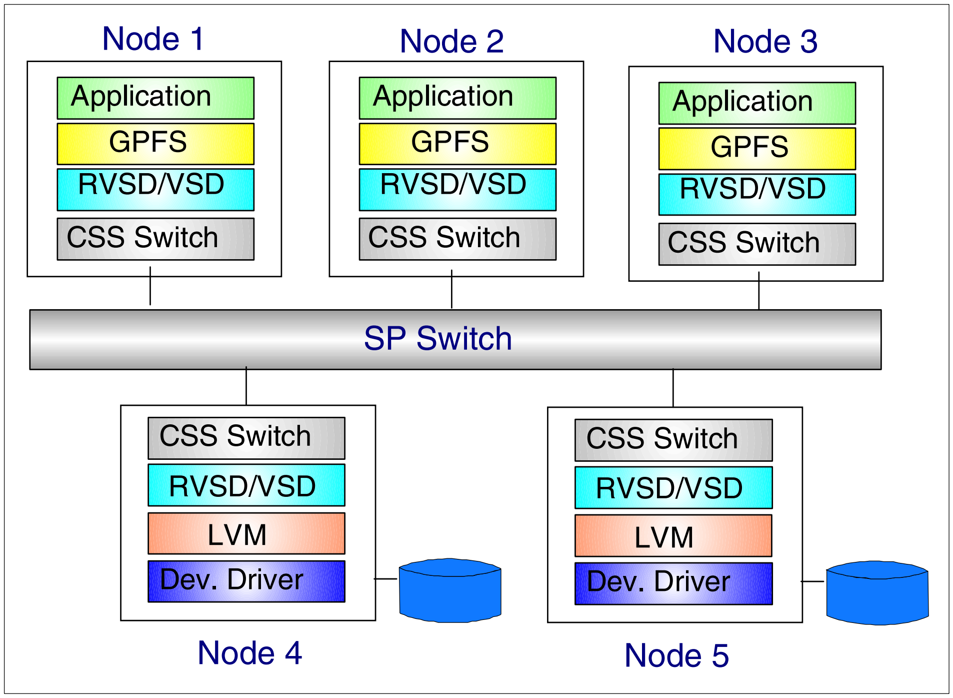

Figure 38. GPFS overview

Nodes 1 to 3 are *application nodes*, which are nodes that have mounted a GPFS file system and are running an application that accesses the file system. Nodes 4 and 5 are *VSD server nodes*, which physically have a number of disk drives attached. GPFS file systems are stored on one or more disks in one or more VSD servers.

A *GPFS nodeset* is a group of nodes that all run the same level of GPFS and operate on the same file system. Application nodes are always in a GPFS nodeset while VSD server nodes may or may not be a part of the GPFS nodeset. In the previous configuration, node 1 to 3 are in GPFS nodeset but node 4 and 5 are not.

Each node may belong to only one GPFS nodeset. There can be more than one GPFS nodes in a system partition. This feature enables you to create a separate nodeset for testing the new level of GPFS code without affecting the nodeset running the production tasks.

A GPFS file system is mounted and accessed on each node in a nodeset as if it were a local file system. To access a GPFS file system across a nodeset or a system partition, you need to mount it like an NFS file system.

14.2 Tuning

GPFS provides simultaneous access to the same or different files across SP nodes with data integrity, high availability, and high performance. To achieve this, GPFS exploits a number of facilities provided by the SP, such as Virtual Shared Disk, Group Services, Switch subsystem, and other components of the AIX operating system.

Tuning GPFS is a complicated and very difficult task. Not only do we need to consider many GPFS parameters, but we need to consider other subsystems that GPFS relies on.

We will now describe the parameters that must be considered for each of the subsystems that GPFS relies on and end with GPFS itself.

14.2.1 Network tuning

Most of the network option parameters, for example, thewall, tcp_sendspace, tcp_receivespace, rfc1323 and so on, are normally already tuned according to the usage type of the node when it is installed. They can be used as the initial values for GPFS.

Here we will discuss *ipqmaxlen,* a parameter that is not normally tuned on the SP nodes.

ipqmaxlen

This parameter specifies the number of received packets that can be queued on the IP protocol input queue. Since both GPFS and VSD software make extensive use of the IP protocol, the default setting of 100 is often insufficient.

If this is set too low, dropped IP packets and consequent performance degradation can result.

As a general guideline, we recommend that ipqmaxlen should be increased to 512 on both VSD servers and GPFS application nodes.

Refer to Section 4.3.2.2, “Networking-related tunables” on page 53 for more e information on how to display and set this parameter.

Note

ipqmaxlen is for IP protocol. If we configure GPFS to use LAPI protocol for the daemons' communication and configure VSD to use KLAPI protocol, this parameter will have no effect on GPFS performance.

14.2.2 Switch adapter tuning

The following parameters, spoolsize and rpoolsize, will severely impact GPFS performance if they are not set correctly.

spoolsize/rpoolsize

The switch send pool is a pinned memory area used by the switch device driver to store the data before sending it over the switch network.

The switch receive pool is a pinned memory area used by the switch device driver to store the data after it receives it from the switch network.

The spoolsize parameter specifies the size in bytes for the switch send pool, and the rpoolsize parameter specifies the size in bytes for the switch receive pool.

It is critical that rpoolsize is set to the maximum size on virtual shared disk servers, or it is very likely there will be dropped packets and unnecessary retries on the VSD client nodes. As a result, the performance will be degraded.

It is also critical that spoolsize is set to the maximum size on VSD servers or there may be a shortage of send pool space, which causes the buddy buffers to be used instead. However, this could result in a situation where all the buddy buffers are used up, which will cause other VSD requests to queue while waiting for the buddy buffer to be available. As a result, the performance will be degraded.

Note

If VSD is configured to use KLAPI, it does not use the switch buffer pool; VSD with KLAPI always uses buddy buffers.

We also recommend setting the two parameters to the maximum on GPFS application nodes unless there is a severe shortage of memory.

Refer to Section 5.2.1, “Switch adapter pools” on page 99 for more information on how to display and set these parameters.

Note

1. For SP switch, the node must be rebooted in order for the changes to take effect. For SP switch2, there is no need to reboot!
1. For SP switch, the maximum size for these pools is 16 MB. For SP switch2, the maximum size is now 32 MB.

14.2.3 SSA adapter tuning

If you are using SSA RAID 5 disk arrays, these parameters max_coalesce and queue_depth may be needed to be adjusted in order to obtain the optimum performance from the disk array.

max_coalesce

This parameter has a high performance impact for sites that are using SSA RAID 5 disk arrays. If the value is set too low, the write performance can be seriously impacted.

max_coalesce is the maximum number of bytes that the SSA disk device driver attempts to transfer to or from an SSA logical disk in one operation.

When there are multiple disk I/O requests in the device driver's queue, it attempts to coalesce those requests into a smaller number of large requests. The largest request (in terms of data transmitted) that the device driver will build is limited by the max_coalesce parameter.

In general, increasing max_coalesce increases the chance that I/O operations for contiguous data are executed as a single operation. This clearly gives performance improvement when the parameter can be set to maximize full stride write (write to all data disks in a disk array) operations.

Ideally, you should set max_coalesce to 64 KB * N (array full stride size) for an N+P RAID array. The default is 0x20000, which corresponds to two 64 KB strips (and applies to a 2+P array). So for an 8+P array, set the value to 0x80000, which is 8x64 KB.

For GPFS, it is recommended that, where possible, the setting of max_coalesce matches with the array full stride size and sets the GPFS file system block size to an integer multiple of the array full stride size.

For example, for an 4+P RAID array, set the max_coalesce to 64 KB * 4 or 256 KB and the GPFS file system block size to 256 KB or 512 KB (2 times the array full stride size) or 1024 KB (4 times the array full stride size). For a 8+P RAID array, set the max_coalesce to 64 KB * 8 or 512 KB and the GPFS file system block size to 512 KB or 1024 KB (2 times the array full stride size).

From the preceding examples, you can see that for certain configurations, such as 6+P, the optimum GPFS file system block size is 64 KB * 6 = 384 KB, 2 * 384 KB = 768 KB, or 3 * 384 KB = 1152 KB. However, all these file system block sizes are not supported by GPFS. Hence, in this case, you should not configure a RAID array for GPFS as 6+P.

Setting max_coalesce smaller than the array full stride size means that several I/O operations have to be initiated in order to read a single file system block.

On the other hand, setting max_coalesce greater than the array full stride size results in degraded write performance for the GPFS file system block size. This is because an additional overhead is incurred, because it has to read the remaining disks in order to compute parity before writing the parity.

Refer to Section 6.3.3.2, “SSA” on page 146 for more information on how to display and set this parameter.

queue_depth

queue_depth specifies the maximum number of commands that the SSA device driver sends to a disk.

This parameter is not so critical to GPFS performance in RAID 5 arrays as the max_coalesce parameter. Nevertheless, having a less than optimum value of queue_depth may mean some of the total I/O bandwidth of the RAID array is wasted.

The default value of queue_depth for an SSA logical disk is 3 for each member disk configured into the array. So, for a 4+P array, the default setting of queue_depth would be 15.

A guideline recommendation is to verify that queue_depth is set to 3*(N+P) for a N+P array. This maximizes the chance of reading data from each component of the array in parallel.

Refer to Section 6.3.3.2, “SSA” on page 146 for more information on how to display and set this parameter.

Note

To change max_coalesce and queue_depth, the device must not be busy. This means that you can only change it when the volume group that the device belongs to is vary offline.

So we recommend that you set the parameters before assigning the disk to the volume group or create any VSD on it.

14.2.4 VSD tuning

In this section, we discuss various VSD tunables and some new features in VSD 3.2 that can provide performance improvements to GPFS (that is, VSD usage of KLAPI and Concurrent VSD [CVSD]).

Refer to Chapter 14, "GPFS" on page 369 for more information about all parameters mentioned in this section and how to display and set them.

VSD_request_blocks

This parameter, sometimes called VSD Request Count, specifies the maximum number of outstanding VSD requests on a particular node.

In VSD 3.2, it is now internally controlled and tuned by VSD subsystem.

VSD_pbufs

This parameter, sometimes called the Read/Write Request Count, specifies the maximum number of outstanding requests that VSD device driver allows for each underlying logical volume.

In VSD 3.2, it is now internally controlled and tuned by VSD subsystem.

VSD_max_buddy_buffer_size

The buddy buffer is a pinned area that VSD server node uses to temporarily store data for I/O operations originated at a remote node. The data in the buffer are purged immediately after the I/O operation completes.

This parameter, the maximum buddy buffer size, specifies the maximum size of a buddy buffer. It should be set to the block size of the largest GPFS file system in the system partition and to the same value on all VSD nodes.

Having the maximum size set lower than the block size of the largest GPFS file system results in the extra overhead for acquiring the additional buddy buffers required to contain a file system block size of data.

Having the maximum size greater than the file system block size results in over allocating memory to the buddy buffer area.

VSD_max_buddy_buffers

This parameter, sometimes called the Number of Max-sized Buddy Buffers, specifies the number of maximum buddy buffer.

It is used to calculate the buddy buffer area by multiplying it to the maximum size of a buddy buffer. For example, if VSD_max_buddy_buffers is 32 and the maximum buddy buffer size is 256 KB, the total buddy buffer area will be 32*256 KB = 8 MB.

Having an insufficient number of buddy buffer can seriously degrade performance because requests for buddy buffers will become queued.

For VSD clients, which do not use the buddy buffer, we recommend setting the maximum number of buddy buffers to 1 (because we can not specify zero for the number of buddy buffers!).

For servers, we recommend the setting be 32 to 96 initially and monitor the value of buddy buffer wait_queue size using the `statvsd` command.

buddy buffer wait_queue size*,* a new output from the `statvsd` command, is the current number of VSD requests waiting in the queue. Its value varies according to the demand for buddy buffers at the time the command runs.

If the output consistently shows a value of n for the buddy buffer wait_queue size, and you are sure that the I/O subsystem is not the bottleneck, you can increase the number of buddy buffers by n.

.

Note

This parameter becomes more important and may need to be increased when you enable VSD to use KLAPI because with IP, if the request is less than maximum IP message size (60 KB), the switch send pool and receive pool will be used. Buddy buffers are used only for requests larger than 60 KB.

However, KLAPI does not use the switch send pool and receive pool. It always uses the buddy buffers.

VSD_maxIPmsgsz

This parameter, sometimes called Maximum IP Message Size, specifies the largest packet size that VSD will send between the client and the server.

Larger values of VSD_maxIPmsgsz will result in fewer packets being required to transfer the same amount of data.

It is recommended to set this parameter to 61140.

Other VSD considerations

Though they are not parameters we can adjust the value, the following things can help improve your GPFS performance.

VSD usage of KLAPI

This is perhaps the most significant performance improvement for GPFS 1.3. Refer to Section 13.3.2, “VSD usage of KLAPI” on page 364 for more detailed information.

Concurrent VSD (CVSD)

By having two active VSD server nodes serving I/O requests for a CVSD, it is possible to gain some performance improvement. Refer to Section 13.3.1, “Concurrent VSD” on page 363 for more detailed information.

Note

Use the following command to create CVSD for GPFS usage:

```
mmcrvsd -c -F descriptorFile
```

The -c option tells the command to create CVSDs. You also must specify two servers in each disk descriptor in the descriptor file.

1 VSD per 1 hdisk

Having too few VSDs configured for the number of physical disks connected to a system can seriously impact performance because the degree of parallelism is reduced.

On the other hand, having multiple VSDs per physical disk can also seriously impact performance because of the increased access contention on the disk.

For GPFS, configuring one VSD per hdisk optimizes GPFS performance.

The `mmcrvsd` command for creating VSD by default creates one VSD on each physical disk specified.

Dedicated VSD server

VSD servers do not have to be dedicated; GPFS and other applications can run on the same nodes as the VSD servers themselves.

However, if possible, we would recommend dedicating the servers, because dedicated VSD servers means it is easier to monitor, control, manage and diagnose performance problems once they happen.

In situations where it is not possible to dedicate VSD servers, it is recommended not to run applications that consume a lot of communication or I/O resources on the VSD server nodes.

14.2.5 GPFS tuning

GPFS performance depends on the proper specification of its parameters. In most cases, the parameters that have the greatest impact on performance are the file system block size and the pagepool.

Some parameters affects specific workloads, for example, maxStatCache. Many parameters are set to a good value by default and do not need to be changed, for example, worker1Thread, worker2Thread, and worker3Thread.

The number of GPFS nodes

Though this is not a tunable parameter (that is, once set it can not be changed later), we include it here since it has a certain performance impact to GPFS if it is underestimated.

When creating a GPFS file system, we need to estimate the number of nodes that will mount the file system. This input is used in the creation of GPFS data structures that are essential for achieving the maximum degree of parallelism in file system operations.

If the input value is not correct, the resulting data structures may limit GPFS' ability to perform certain tasks efficiently, for example, the allocation of disk space to a file.

If you cannot anticipate the number of nodes, allow the default value of 32 to be applied. However, in case there is a possibility that the number of nodes that mount the file system will exceed 64 (or 128), specify 64 (or 128).

File system block size

Although file system block size is a parameter that cannot be changed easily (to change it means rebuilding the file system, which usually requires a lot of effort), we include it here because it is one of the most important parameters that can affect the performance of GPFS. It is worth consideration before deciding on the proper value.

The file system block size is specified at the file system creation time via the -B option in the `mmcrfs` command.

GPFS 1.3 offers two more block sizes, 512 KB and 1024 KB, in addition to the usual 16 KB, 64 KB, and 256 KB.

The 256 KB block size is the default and normally is the best block size for file systems that contain large files accessed in large reads and writes.

The 16 KB block size optimizes use of disk storage at the expense of larger data transfers and is recommended for applications having a large numbers of small files.

The 64 KB block size offers a compromise. It makes more efficient use of disk space than 256 KB, which allows faster I/O operations than 16 KB.

The 512 KB and 1024 KB block size may be more efficient when the data accesses from the applications are larger than 256 KB, for example, technical and scientific applications doing very big block I/O operations.

Note

If you create a file system with block sizes of 512 KB or 1024 KB, you will only be able to access the information through GPFS 1.3, because prior releases of GPFS do not support these block sizes.

Besides the block size of data accessed by the application and the number and size of the files in the file system, the use of RAID devices also needs to be considered when determining the block size. With RAID devices, a larger block size may be more effective and help you avoid the penalties involved in small block write operation to RAID devices.

For example, in a RAID configuration utilizing 8 data disks and 1 parity disk (a 8+P configuration) which utilizes a 64 KB stripe size, the optimal file system block size would be 512 KB (8 data disk * 64 KB stripe size). A 512 KB block size would result in a single data write that encompassed the 8 data disks and a parity write to the parity disk.

If we use a block size of 256 KB, write performance would be degraded. A 256 KB block size would result in a four-disk writing of 256 KB and a subsequent read from the four remaining disks in order to compute the parity that is then written to the parity disk. This extra read degrades the overall performance.

The current file system block size can be identified by executing the `mmlsfs` command. For example:

```
root@sp6en0:/: mmlsfs gpfsdev -B
mmrts: Executing "mmlsfs /dev/gpfsdev -B" on node sp6n05
flag value          description
---- -------------- -----------------------------------
 -B  262144         Block size
```

pagepool

pagepool specifies the amount of pinned memory reserved for caching file data, directory blocks and other file system metadata, such as indirect blocks and allocation maps.

pagepool is one of the most important parameters that has a high impact on GPFS performance because it allows GPFS to implement read and write requests asynchronously via the read ahead and write behind mechanisms. It also helps increase performance by allowing the reuse of cached data in the pagepool.

Setting pagepool too high could impact overall systems performance because of the reduced real memory available to applications.

On the other hand, setting pagepool too low could seriously degrade the performance of certain types of GPFS applications, for example, ones that do large sequential I/O operations.

A sufficiently large pagepool is necessary to allow an efficient read prefetch and deferred write.

For applications that are doing large amounts of sequential read or write, or are frequently re-reading various large sections of a file, increasing pagepool from the default value significantly improves performance.

The setting of pagepool can also be particularly critical for applications which do random I/O. For random read I/O, the benefits of a larger pagepool are significant, because the chances a block is already available for read in the pagepool is increased. Similarly, the performance of random writes will also benefit if pagepool space is not constricted. This is because applications might otherwise have to wait for pagepool space to be freed while dirty buffers are flushed out to disk.

The pagepool parameter can be set to between 4 MB and 512 MB per node. The default setting is 20 MB.

The pagepool size may be reconfigured dynamically with the `mmchconfig -i` command. For example, to change pagepool size on all nodes in the GPFS nodeset 1 to 40M:

```
root@sp6en0:/: mmchconfig pagepool=40M -C 1 -i
mmchconfig: Command successfully completed
mmchconfig: Propagating the changes to all affected nodes.
This is an asynchronous process.
```

Please note the value must be suffixed with the character "M", and the -i flag (introduced in GPFS 1.2) makes the change effective immediately without the need to stop and restart GPFS daemons.

Because the setting of the pagepool parameter is so specific to the number of I/O requests being generated from a node and the application I/O pattern, this is one parameter that is worthwhile assessing on a per node basis.

Further, because it can be modified dynamically, changes to pagepool could be implemented around GPFS applications. For example, a run of a specific application sequence that would benefit from an increased pagepool could have commands that increase the pagepool and reset it back to the original values wrapped around the main application script.

Moreover, it would be relatively easy to repeat consecutive runs of an application with a gradually incrementing pagepool to ascertain the minimum level of pagepool necessary to deliver the optimum I/O throughput.

maxFilesToCache

MaxFilesToCache specifies the total number of different files that can be cached at one time.

Every entry in the file cache requires some pageable memory to hold the file’s i-node plus control data structure. This is in addition to any of the file’s data and indirect blocks that might be cached in the pagepool area.

Setting it too high may not improve application performance but actually degrade it. On the other hand, setting it too low could result in performance penalties for applications that make use of a large number of files.

The maxFilesToCache parameter can be set to between 1 and 100,000. The default setting is 1000.

This parameter should be set on each node to the maximum number of files that will be accessed concurrently in the GPFS file system from a particular node.

maxFilesToCache can be modified with the `mmchconfig` command. For example, to change the maxFilesToCache parameter to 1500 on all nodes in the GPFS nodeset 1, use the following:

```
root@sp6en0:/: mmchconfig maxFilesToCache=1500 -C 1
mmchconfig: Command successfully completed
mmchconfig: Propagating the changes to all affected nodes.
This is an asynchronous process.
```

Since this change is not dynamic, the GPFS subsystem must be stopped and restarted for the change to take effect.

maxStatCache

maxStatCache specifies the number of entries in additional pageable memory used to cache attributes of files that are not currently in the regular file cache. This is useful to improve the performance of both the system and GPFS stat() calls for applications with a working set that does not fit in the regular file cache.

The maxStatCache parameter can be set to between 0 and 1,000,000. The default setting is 4 times the maxFilesToCache.

For systems where applications test the existence of files or the properties of files without actually opening them (for example, backup applications like TSM usually does), increasing the value of maxStatCache can significantly improve the performance.

maxStatCache can be modified with the `mmchconfig` command. For example, to change the maxStatCache parameter to 4000 on node 10 which is a TSM server, use the following:

```
root@sp6en0:/: dsh -w sp6n10 mmchconfig maxStatCache=3000
sp6n10: mmchconfig: Command successfully completed
sp6n10: mmchconfig: 6027-1371 Propagating the changes to all affected nodes.
sp6n10: This is an asynchronous process.
```

Notice that since we only change the parameter on node 10, we use dsh to send the command to run on node 10. Also notice that since node 10 can belong to only one GPFS nodeset, there is no need to specify -C option on the command.

prefetchThread
prefetchThreads is a parameter that specifies the maximum number of threads that GPFS uses for the read ahead operation.

The minimum value for prefetchThreads is 2 and the maximum, which is the default, is 72.

It is recommended not to change this parameter.

worker1Thread
worker1Thread is a parameter that specifies the maximum number of threads that GPFS uses for the write behind operation.

The minimum value for worker1Threads is 1 and the maximum, which is the default, is 48.

It is recommended not to change this parameter.

worker2Thread
worker2Thread is a parameter that specifies the maximum number of threads that GPFS uses to exercise control over directory access and other miscellaneous I/O operations.

The minimum value for worker2Threads is 1 and the maximum, which is the default, is 24.

It is recommended not to change this parameter.

worker3Thread
worker3Thread is a parameter that specifies the maximum number of threads that GPFS uses for the i-node prefetch operation.

The minimum value for worker3Threads is 1 and the maximum, which is the default, is 24.

It is recommended not to change this parameter.

Note

It is recommended not to change these *Thread parameters because GPFS internally controls the number of these threads and respawns as needed up to the limit specified by the corresponding parameter.

Having the default as the maximum value works well in almost all cases. Only in rare circumstances, for example, when your I/O configuration cannot cope with the workload, would you need to consider adjusting it down.

comm_protocol

In GPFS 1.3, the communication protocol between the GPFS daemons within a nodeset can now be selected between TCP/IP or LAPI (Low-Level Application Programming Interface).

This choice can be made when first configuring GPFS (the `mmconfig` command) or when changing configuration parameters (the `mmchconfig` command). The default is TCP.

How much performance improvement you receive by enabling LAPI depends on the I/O access patterns and the types of your application. For example, if your application reads/writes a lot of small files, there will be a lot of token usage and hence a lot of communication traffic between Token Managers and the Token Manager server. In this case, it is likely that you will gain significant performance improvement.

It is also important that you understand what is required when choosing LAPI as the communication protocol. A switch adapter window must be reserved on each node in the GPFS nodeset.

To successfully reserve an adapter window for use by GPFS, all LoadLeveler jobs must be stopped prior to issuing either the `mmconfig` or the `mmchconfig` command, because if LoadLeveler is not stopped, it will reserve all adapter windows for its use. If an adapter window cannot be reserved on each node for use by GPFS, the command fails.

After reserving an adapter window on each node, the window can not be used by any application (even LoadLeveler) until it is released.

The adapter window is released only when the communication protocol is changed back to TCP or the node is deleted from the GPFS nodeset.

For more detail information regarding adapter windows, see the *PSSP: Administration Guide*, SA22-7348 and refer to the section on Understanding Adapter Window.

To configure GPFS to use LAPI communication, we need to stop GPFS, use `mmchconfig` to enable LAPI, and restart GPFS:

```
root@sp6en0:/: dsh stopsrc -s mmfs

root@sp6en0:/: mmchconfig comm_protocol=LAPI -C 1
Verifying GPFS is stopped on all nodes ...
mmchconfig: Successfully changed communication protocol to LAPI
mmchconfig: Propagating the changes to all affected nodes.
This is an asynchronous process.

root@sp6en0:/: dsh startsrc -s mmfs
```

Another point that you need to be aware of is that after setting the communication protocol to LAPI, GPFS must be stopped and restarted when there are certain configuration changes, for example, when you add or delete nodes.

This is due to the fact that there will be changes in the routing table on the adapter window when you add/delete nodes and the switch adapter window loads the route tables only when GPFS initializes.

Other GPFS considerations

Below are some other GPFS considerations.

data/metadata

It is possible to designate individual VSDs within a particular GPFS file system to contain data only, metadata only, or both data and metadata. The default is to use all VSDs for both data and metadata.

The idea of storing data and metadata on a separate set of disks seems attractive at first, because metadata traffic tends to consist of relatively small random reads and writes and it is quite different from the large sequential I/Os of the typical GPFS application. Moreover, with the increasing use of RAID devices to store the data, separating the metadata seems to be a good idea in order to avoid the small block write penalty of RAID device.

However, if metadata is to be stored on a separate set of VSDs, we know, from lab experience, that in order to prevent these set of disks from becoming a bottleneck, it seems that we would need almost as many disks to store the metadata as to store the data! This is generally impractical for most installations.

Availability considerations also strengthen the argument for interspersed data and metadata. For example, isolating metadata on non RAID disks means that if those disks fail, there will be little hope of repairing file system consistency unless they are replicated (we recommend you always replicate metadata). The advantages of storing data on RAID arrays configured with hot-spare disk will be compromised if they rely on non RAID disks for storing metadata.

Thus, we recommend adopting a general guideline of interspersing data and metadata across VSDs and always replicate metadata.

File System manager location

One GPFS node for every GPFS file system has the responsibility for performing File System manager services. The File System manager is responsible for:

- Administering changes to the state of the file system
- Repairing the file system
- Administering file system allocation requests
- Administering file system token requests

When you configure GPFS with the `mmconfig` command, you can specify a list of nodes to be included in the GPFS nodeset with the -n filename option.

Each line in the file contains the switch hostname and whether or not this node should be included in the pool of nodes from which the File System manager node is chosen.

The default is to have the node included in the pool. Specify <switch hostname>:client if you don't want this node to be a File System manager.

Note

This method is recommended over modifying the individual /var/mmfs/etc/cluster.preferences file, because it changes the preference file on all nodes in the GPFS nodeset for you.

To identify the File System manager, use the `mmlsmgr` command. For example:

```
root@sp6en0:/: mmlsmgr -C 1
mmrts: Executing "mmlsmgr " on node sp6n05
file system      manager node
---------------- ------------------
gpfsdev          5 (sp6n05)
```

The `mmlsmgr` command displays the File System manager node for all file systems; in the preceding example, node 5 is the File System manager for `gpfsdev`.

Normally, the configuration manager attempts to balance File System Managers by selecting a node that is not currently a File System Manager for any other file system. However, since the Token Manager Server for a file system resides on the same node as the File System manager, if there is a lot of I/O activity going on in that file system that requires a lot of token usage, this situation can affect the performance of other applications running on that node. Thus, in certain circumstance, you might want to move the File System manager to another node.

In prior releases, you can influence GPFS' decision to choose File System manager by listing the nodes in the /var/mmfs/etc/cluster.preferences file; you cannot specify exactly which node will become the File System manager for a file system. This has been changed in GPFS 1.3. A new command, `mmchmgr`, is provided to facilitate the designation or change of the File System manager location.

For example, to change the File System manager of `gpfsdev` device to node 7:

```
root@sp6en0:/: mmchmgr gpfsdev sp6n07
mmrts: Executing "mmchmgr /dev/gpfsdev sp6n07" on node sp6n05
GPFS: 6027-628 Sending migrate request to current manager node 5.
GPFS: 6027-629 Node 5 (sp6n05) resigned as manager for gpfsdev.
GPFS: 6027-630 Node 7 (sp6n07) appointed as manager for gpfsdev.
root@sp6en0:/: mmlsmgr -C 1
mmrts: Executing "mmlsmgr" on node sp6n05
file system      manager node
---------------- ------------------
gpfsdev          7 (sp6n07)
```

Metadata manager location

There is one metadata manager for each open file in the GPFS file system. It is responsible for the integrity of the metadata of that file. GPFS selects the node that first opens that file to be the metadata manager.

In some case, for example, when a file is accessed by multiple nodes and those nodes are heavily stressed, there may be some merit in considering opening that file first from a relatively quiet node so that it will be the metadata manager for the file.

Non-tunable parameters

Table 39 shows the parameters that were previously tunable but have been changed in GPFS 1.3.

Table 39. Parameters that are now internally controlled by GPFS

Parameter	Value
mallocsize	Calculated from maxFilesToCache
indirect block size	4 KB
i-node size	512 bytes
GPFS daemon priority	40

14.3 Summary of GPFS tuning recommendations

Table 40 summarizes the tuning recommendations for the GPFS subsystem:

Table 40. Summary for tuning a GPFS subsystem

Parameter	Recommended setting for GPFS
ipqmaxlen	512
spoolsize	16 MB for SP switch, 32 MB for SP switch2
rpoolsize	16 MB for SP switch, 32 MB for SP switch2
max_coalesce	Matches with the GPFS file system block size
queue_depth	3 * the total number of disk in RAID array
request blocks	Internally controlled by VSD
pbufs	Internally controlled by VSD
maximum buddy buffer size	Matches with the GPFS file system block size
number of buddy buffer	32 to 96 initially, monitor and adjust as needed
maximum IP message size	61140
GPFS block size	Depends on the application; default is 256 KB
pagepool	Depends on the application; default is 20 MB
maxFilesToCache	Depends on the application; default is 1000

Parameter	Recommended setting for GPFS
maxStatCache	Depends on the application; default is four times of maxFilesToCache
prefetchThread	Do not change this parameter
worker1Thread	Do not change this parameter
worker2Thread	Do not change this parameter
worker3Thread	Do not change this parameter
comm_protocol	Set to LAPI, if possible

Chapter 15. Web applications

In this chapter, we will show you how an SP System can be used as a WebServer and how several products and their tuning capabilities can affect the SP System. Most of the tuning actions are AIX-related, so there is no real need to focus too much on SP related settings. We are not going into deep detail because it would be too much information for this book (you can refer to *Websphere V3 Performance Tuning Guide*, SG24-5657 for more information). There are other applications, like ZEUS WebServer, that are also available for RS/6000 AIX machines; you can also check `http://www.zeus.com` for more information on these applications.

15.1 Tuning web applications on the SP

Today, we have a huge growing Internet market. Every day, millions of Internet Users visit websites. They are looking for news, doing some shopping, and much more. Hardware performance increases about 60 percent a year. The amount of users increases more than 90 percent a year and the traffic almost 1000 percent a year. That leads us to a point where application tuning is needed more than ever.

The 2000 Olympics in Sydney is an excellent example of the huge amount of traffic that can occur all at once. The official Olympic homepage (`http://olympics.com`) got more than 11 300 000 000 (11.3 billion) hits during the games. That is more than 1.2 million hits per minute. Thus, we need to look at several points.

One part of application tuning include Web Server Applications like IBM WebSphere or the ZEUS WebServer.

15.1.1 Websphere - an overview

WebSphere Application Server V3 is one application from IBM that can be used to build a Internet and/or Intranet sites. It is offered in three editions:

- WebSphere Application Server, Standard Edition
- WebSphere Application Server, Advanced Edition
- WebSphere Application Server, Enterprise Edition

What is tunable now in Websphere? What is the focus? Here are some checkpoints to look at:

- The hardware capacity and settings

- The operating system settings
- The Web application settings

In general, upgrading the hardware is one step toward increasing performance. We will not describe hardware related settings and AIX settings too deeply, but we will give information on where to look at when using a WebServer application.

15.1.2 What can be tuned?

There are many parts to tune in a WebServer installation:

- AIX TCP/IP Tuning
- WebSphere Engine
- Database tuning (Oracle,DB2 and more)

> **Performance tip**
>
> You can improve performance when you run the WebSphere Application Server and the database server on separate servers.

There are four fundamental decision points that have their own tuning options:

1. The Web server (HTTP server)
2. The WebSphere Application Server Engine
3. The Java Virtual Machine (JVM)
4. The database server

Take a look at Figure 39 on page 395 for a visualization of these decision points.

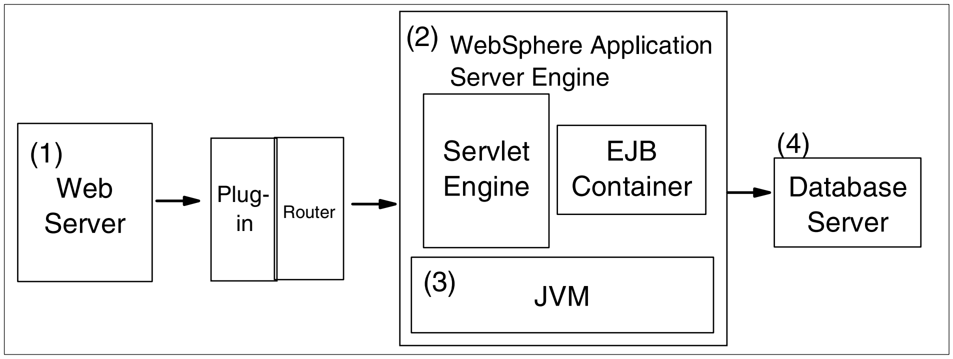

Figure 39. Fundamental tuning points of WebSphere application server settings

Refer to the following redbooks for particular tuning methods:

- *Understanding IBM RS/6000 Performance and Sizing*, SG24-4810
- *WebSphere V3 Performance Tuning Guide,* SG24-5657

15.1.3 AIX tunables for web related applications

The tunable parameters from the `no` command give us many ways to tune specific environments. See Table 41 on page 396 for a summary of what is related to WebServer tunables:

Table 41. Tunable no command parameters

Parameter	Default Value	tuned Value [a]
sb_max	65536	262144
tcp_sendspace	4096	28000
tcp_recvspace	4096	28000
tcp_timewait	1 (15 seconds)	5
somaxconn	1024	8192
nbc_max_cache	131072	100000
nbc_pseg_limit [b]	0	10000
nbc_pseg [c]	0	20000
nbc_limit	0	786432

a. These values are results were taken from IBMs data sheets at `http://www.spec.org. Look for` SPECWeb96.
b. New parameter available with AIX 4.3.3 and later.
c. New parameter available with AIX 4.3.3 and later.

For a detailed description of the `no` parameters refer to Section 4.3.2, “AIX tunables” on page 46.

Note

There is no ALL FOR ONE tuning solution; this is only a sample that can help to improve performance.

15.1.4 Performance test tools for webserver

There are some special tools that simulate the load on a web server. We will give you an overview of what is available and where to get it:

- WebStone - Available at `http://www.mindcraft.com`

  WebStone is an open source benchmark that is available as freeware from Mindcraft. Originally developed by Silicon Graphics, WebStone measures the performance of Web server software and hardware products. Webstone simulates the activity of multiple Web clients, thus creating a load on a web server. The load can be generated from either one client computer or by multiple client computers (up to 100 simulated clients on a single computer!). Because WebStone is freeware, it does not offer many of the features available from other products.

- Apache Bench and JMeter - Available at `http://www.apache.org`

  The IBM HTTP Server (IHS), which is included with WebSphere Application Server, does include the Apache Bench (AB) tool on UNIX platforms. Apache Bench is Perl script-based and allows the simulation of a HTTP client load.

  JMeter is another freely available tool from the Apache Software Foundation. JMeter is a Java desktop application designed to test URL behavior and measure performance. Apache JMeter may be used to test server performance both on static and dynamic resources (files or CGI, servlets, Perl scripts). Simple to use, JMeter is limited to 20 concurrent requests per JMeter client, but can be used for initial performance testing.

- Rational Suite Performance Studio - Available at `http://www.rational.com`

  Performance Studio offers support for a variety of clients, both Web and non-Web based. Among the clients supported are HTML, DHTML, Document Object Model, Visual Basic, Visual C++, Java, ActiveX, and PowerBuilder. Performance Studio records user inputs for playback as scripts that are used in performance testing.

- WebLoad - Available at `http://www.radview.com`

  WebLoad from Radview provides support for the HTTP 1.0 and 1.1 protocols, including cookies, proxies, SSL, keep-alive, and client certificates. Support is also provided for a variety of authentication mechanisms, such as basic authentication, proxy, form, NT challenge response, client certificate, and user agent.

- LoadRunner - Available at `http://www.merc-int.com`

  LoadRunner from Mercury Interactive Corporation supports a variety of different clients, including Web, ERP, database (DB), Java or remote terminal emulator (RTE).

Information for IBM employees only

In addition to the tools discussed in this section, AKtools is available for IBM employees only. AKtools is a set of internal IBM applications which allow you to test Web application performance. It includes AKstress and AKrecord. It is capable of simulating hundreds or even thousands of HTTP clients. See the following URL for more information:

`http://events.raleigh.ibm.com/aktools/docs/index.html`

15.1.5 Monitoring tools

As with any other AIX based System, we can still monitor many instances with our already known tools: `ps`, `svmon` and many others mentioned here and in Chapter 10, "Resource monitoring" on page 203 and Chapter 11, "Tools" on page 243. If you need more information, you may refer to these chapters. Additionally, the Websphere Application Server Resource Analyzer provides a number of summary and discrete monitoring functions for a variety of resources.

Figure 40 gives you a short look at what the Resource Analyzer output can look like.

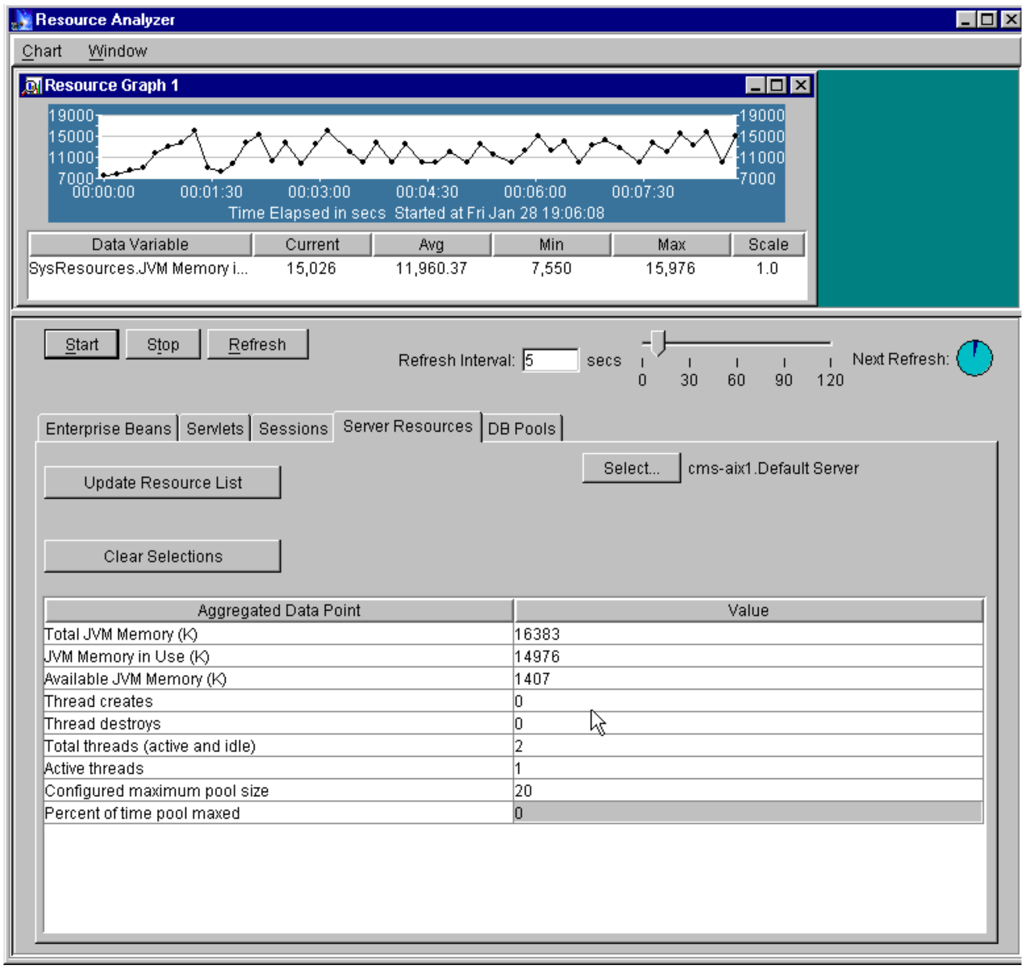

Figure 40. Resource Analyzer with case1 (-mx64m)

You have a lot of values you can monitor with the tool. It is much easier to analyze your WebServer performance when you are able to make those plain numbers visible (the values provided from AIX or the application). However, this not be a description on how to use it; for more information on this subject, refer to the *WebSphere V3 Performance Tuning Guide*, SG24-5657.

Chapter 16. DB2 Universal Database

Relational Database Management Systems (RDBMS) have become more and more the core IT systems of companies and large enterprises. High performance of these systems is often vital for the success of a company, and because the systems get bigger every year, the issue gets bigger as well.

Database performance is a large area with many different aspects. It is dependent on a large number of factors, such as the hardware, the applications, the workload, the layout of the disk subsystem, and an uncounted number of system and database parameters.

Today, the scalability needed to handle large volumes of data capacity is increasing at a rapid pace. As such, the RS/6000 SP system's enormous data capacity and scalability make it a very good choice for businesses facilitating high volumes of data. Its parallel processing capabilities enhance performance and throughput.

The architecture of the RS/6000 SP system, together with the DB2 UDB database, provide a high-performance and scalable platform for any growing data application.

This chapter will give some general ideas about DB2 UDB configuration in the RS/6000 SP system and the best way to take advantage of its parallel structure. We will also describe the last TPC-H record reached by IBM in benchmarking with DB2 UDB and SP systems. This is a very good example of tuning for performance that can be used as reference, but do not forget it is only an special case; each particular database and system should be configured and tuned depending of the own characteristics of them.

For complete information about DB2, refer to the following:

- *DB2 UDB Administration Guide: Performance V6*, SC09-2840
- *DB2 UDB System Monitor Guide and Reference V5*, S10J-8164

For detailed information about AIX, refer to the following:

- Understanding IBM RS/6000 Performance and Sizing, SG24-4810

Also, check these IBM redbooks:

- *Database Performance on AIX in DB2 UDB and Oracle Environments*, SG24-5511
- *RS/6000 Performance Tools in Focus*, SG24-4989

• *RS/6000 Performance Tools in Focus*, SG24-4989

16.1 General concepts about parallel databases

Large and parallel databases benefit from certain system architectures, such as shared memory, shared disk, or shared nothing architectures. The implementation of parallel databases also depends on the hardware architecture on which the database system runs.

In a *shared nothing* environment, every CPU has its own memory and its own set of disks and, therefore, does not have to compete for resources with other processors (see Figure 41). These *loosely coupled* systems are linked by a high speed interconnection. This environment is referred to as Massively Parallel Processors (MPP). An implementation of this concept is the RS/6000 SP system.

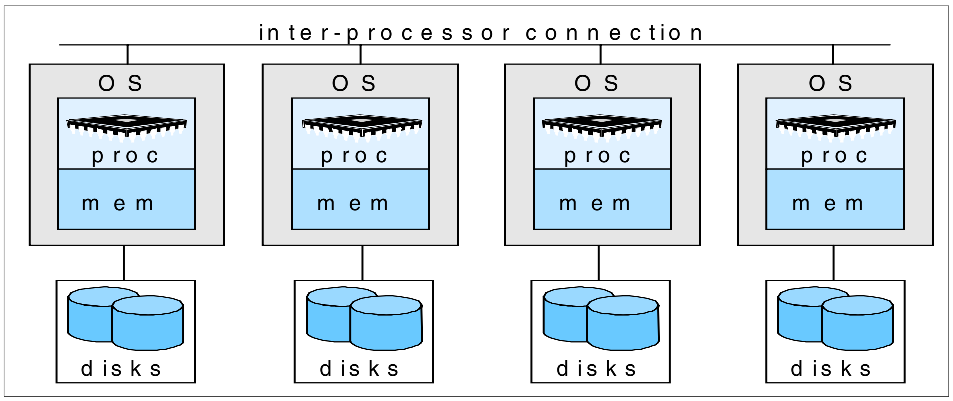

Figure 41. Shared nothing

IBM implementation of parallel database concepts on AIX is DB2 Universal Database Enterprise - Extended Edition (DB2 UDB EEE). This product is based on the same code as DB2 UDB Enterprise Edition and extended with the fileset *DB2 UDB Parallel Extension* that enables the distribution of data and indexes over multiple database partitions. For this reason, it is also called a *partitioned database system* or *clustered system*.

On an RS/6000 SP, you may decide to assign one partition to every single node of your system, but it is also possible to declare several partitions per node.

On each database partition, one database manager is responsible for a portion of the database's data. Each database server has its own set of data.

The fact that data is partitioned across database partition servers is transparent to users and applications.

DB2 UDB EEE executes everything in parallel. All database functions, such as SELECT, INSERT, UPDATE, and DELETE, are performed in parallel on all database partitions. Both database activities, such as data scan, index scan, joins, sorts, index creation, or table reorganization and DB2 UDB utilities, such as data load, backup, and restore, are executed simultaneously on all partitions. The loading of very large amounts of data can be performed much faster in this parallel database environment than on a serial database system.

Communication across all database nodes is realized by the *Fast Communication Manager* (FCM). Each database partition has one FCM daemon to provide communication support in order to handle DB2 UDB agent requests and to manage message buffers. In an SP environment, the FCM daemons interact over the SP Switch and TCP/IP sockets.

16.1.1 Inter-partition and intra-partition parallelism

DB2 UDB EEE architecture employs several parallel query techniques to exploit all available system resources for query execution:

- *Inter-partition* parallelism means that the function is executed in parallel by each database partition. An example would be when a user or application issues an SQL statement, such as a SELECT statement, to fetch data with certain conditions from many tables spread over multiple nodes. In this case, the coordinator node sends this request to all database partitions. The database manager on each node selects the data from tables stored on the disks, sorts the data, and sends all rows that meet the selected conditions back to the coordinator node. On this node, all rows are finally merged and returned to the user or application. In this example, the function (query) is shipped to all nodes, and only the data that satisfies this request is sent back across the network. This concept reduces the network traffic and is known as *function shipping.*
- *Intra-partition* parallelism allows different operators in the same query to be executed in parallel by the same database partition node. If, for instance, an application performs a query including a SCAN, a JOIN, and a SORT, the database manager can execute this request in parallel depending on the setting of dedicated DB2 UDB configuration parameters.

The DB2 UDB cost-based optimizer decides whether a user statement runs in parallel or not, which stages of that user statement are parallelized, and how these stages are parallelized.

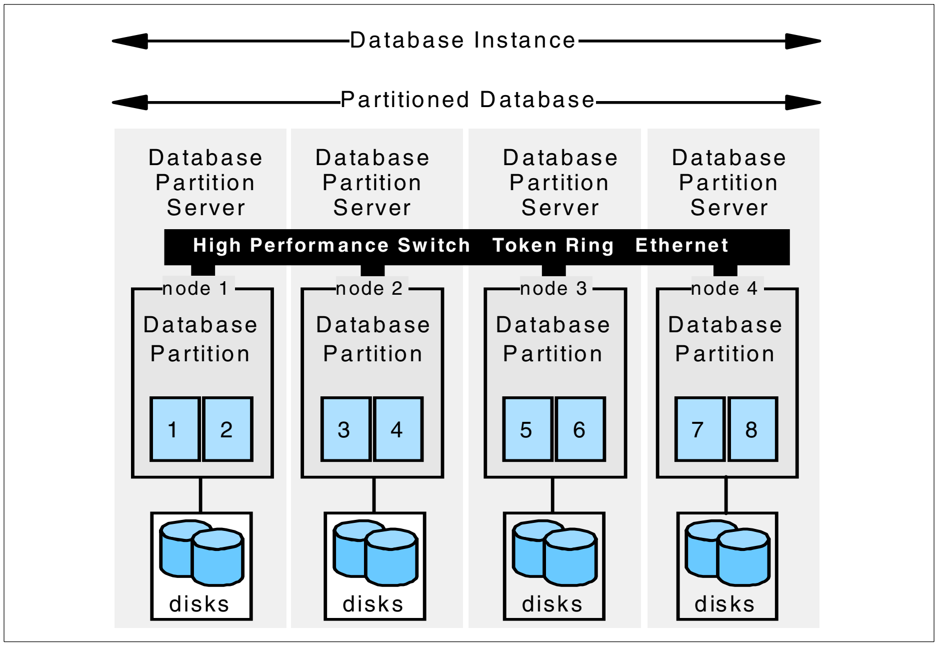

Figure 42. DB2 UDB EEE structure

16.1.2 Database partitioning

DB2 UDB EEE employs a flexible topology for data storage to achieve a high degree of parallelism and performance. A database is set up to have a certain number of *database partitions* - where each partition is a subset of a database containing its own user data, indexes, transaction logs, and configuration files.

Database partitioning is controlled via *node groups* that are defined to the system. A node group is an object defining the number of partitions to be used. Within the node group, tablespaces are defined and all tables are created. A partition map exists for each node group defined to the database.

The partition map defines a specific database partition for each of a possible 4096 values. Each time a row is added to the database, the value of the partitioning key (a column or a set of columns in the table defined as the partitioning key) is hashed, resulting in a value between 0 and 4095. The row is then added to the corresponding partition.

For each tablespace, one or more containers (files or devices) are identified into which table data and indexes will be stored. By specifying more than one container per tablespace in each database partition, I/O parallelism can be employed by the data manager for enhanced performance benefits.

The DB2 UDB EEE partitioning mechanism provides the following advantages:

- Data is automatically distributed across multiple database partitioning, which can be physically spread across independent servers, thereby enabling databases that scale into the terabyte range.
- DB2 UDB EEE is aware of the partitioning scheme in the data definition language (DDL), data manipulation SQL, and at run time. The DB2 UDB EEE optimizer takes advantages of partitioning knowledge to evaluate the costs of the different operations and thus choose the optimal execution strategy for a given SQL statement.
- In the event that there is data skew after the initial distribution, DB2 UDB EEE utilities can automatically analyze and correct the skew.
- Incremental database growth is supported by adding partitions to the database, changing the partition map to include these new database partitions, and automatically redistributing the data.
- Applications can call an API to determine the location of a row, and can route the transaction directly to the node where the row is located. This API can also be used by transaction processing applications, such as CICS or Encina module (included in *Websphere Application Server Enterprise Edition V3.0,* G325-3920) to route transactions to the appropriate node and thus improve performance.

16.2 Tuning database systems for performance

As a general recommendation for tuning methodology the first step is document the current system, carefully study the problems one by one, and then change it. This must be done in both sides (system and database).

16.2.1 Document the current system

Before changing anything on the system or database, it is worth checking a few basic parameters to make sure that they are either the default value (a sensible number) and that we know when, by whom, and why they have been changed. Many times default values are not the best setting and differ for standard recommendations. So if they are changed, try to document the reasons for that.

You should have an understanding of the following topics:

- **Machine details**
  - CPU rating and number of CPUs
  - Memory size and type
  - Disk types and configuration
  - Disk speeds in seek time and throughput
  - Disk usage and logical volume
  - Adapter types and ratings
- **Workload details**
  - The busy periods of the system so you can tune for these peaks in workload.
  - The number of users logging on and actually being busy during the peak.
  - The number of transactions per minute during the peaks.
  - If there are online and batch peaks in the workload.
- **AIX virtual memory parameters**

  Check that all the values are set to their default values. Use the `vmtune` command for that. See Section 11.1.38, “vmtune” on page 330 for more details.
- **AIX system tunable parameters**

  Use the `lsattr -E -l sys0` command to detail the AIX tunable parameters.
- **Network parameters**

  Use the `no -a` command to document the network parameters. See Section 4.3, “AIX network tunables” on page 45 for more details.
- **Hardware configurations**

  Use the `lscfg` command to document the adapters.
- **Document the RDBMS parameters**

  Use the DBA tools to output the database parameters currently in use:

  ```
  > db2 get dbcfg for <database>
  > db2 get dbmcfg
  ```

  See IBM DB2 documentation for more details about these commands.

- **Check for errors**

  The first thing is to check is if the system is, in fact, trying to tell you that there is a problem. In AIX, the first place to look is in the AIX system error log. To do this, use the commands `errpt | pg`. Second, check the RDBMS error logs; on a DB2 UDB system, these are located in the $DIAGPATH/db2diag.log file. There may be only one file or one per database partition, depending on the configuration.

- **Upgrade to the latest fix levels**
  - For AIX - PTF, access the following:

    `http:/services.software.ibm.com/support/rs6000`
  - For hardware firmware, access the following:

    `http:/www.rs6000.ibm.com/support/micro`
  - For RDBMS fixes or PTF, access the following:

    `http:/www.ibm.com/software/data/support`
  - Application fixes or newer versions

16.2.2 Bottlenecks, utilization, and resources

The first thing to investigate is the utilization level of each of the system resources (for example, CPU, system memory, database memory, and network) and compare these to the levels we would like to find. Each of these resources has an optimal level which, if exceeded, has a negative impact on the overall system performance. These levels differ between workloads, and various people have different opinions on these levels as well.

You can use Table 42 as a starting point for tuning.

Table 42. Bottleneck thresholds depend on the resource and workload

Resource	Measured by	OLTP	DSS	OLAP	Batch
CPU	Percent system + percent user time	70 percent	80 percent	70 percent	100 percent
System memory	See [1]	99 percent	99 percent	99 percent	99 percent
RDBMS memory	Cache and library hit ratio	99 percent	80 percent	99 percent	60 percent
Adapters	Percent busy and throughput	50 percent	50 percent	50 percent	50 percent [2]
Disks	Percent busy	40 percent	40 percent	40 percent	60 percent [2]

Resource	Measured by	OLTP	DSS	OLAP	Batch
Network	Transfers and throughput	30 percent	40 percent	30 percent	60 percent [2]
Notes: [1] AIX makes use of all available memory once it is running for any length of time. [2] Batch can stress the system higher than these levels, but checks need to be made in order to make sure that other workloads or users of these resources are not affected. OLTP: Online Transaction Processing DSS: Decision Support Systems OLAP: Online Analytical Processing					

Generally speaking, batch and DSS workloads can use higher utilization levels because they are focused on throughput rather than response time.

In benchmark scenarios, better performance is reached by pushing the CPU close to 100% busy with this distribution:

- DSS: 80% user - 20% kernel
- OLTP: 70% user - 30% kernel

But this is only possible when running very well tuned applications.

16.2.3 Insufficient CPU and latent demand

On well performing machines with many users attached, the workload varies during the working day. The typical pattern is where there is a dip in workload during the lunch time period, and the CPU is not 100 percent busy during the peaks in mid-morning and mid-afternoon.

On an overworked system, the picture changes quite a lot, because during the peaks, the CPU becomes 100 percent busy. It is nearly impossible to work out how much CPU power is required to stop the CPU from becoming the bottleneck. If the system is then tuned, the CPU may still be the bottleneck, even though the tuning might improve the performance on the machine.

In these cases, the response times may improve even if the CPU is still overworked. If the CPU is still overworked after tuning, and an upgrade is recommended, it is still going to be very hard to estimate the CPU requirements of the upgraded machine because the height of the peaks can not be determined.

16.2.4 Insufficient memory

When a machine does not have sufficient memory, the symptoms can be difficult to clearly detect. First, the machine might look like it has a disk or I/O throughput problem. This can be caused by simple UNIX paging and swapping activity. AIX commands, such as `vmstat`, can highlight paging activity. As the RDBMS buffer cache or pool takes up memory, one way is to reduce its size in order to free memory. This may stop paging but may mean the database cannot keep enough of the database in memory for high performance, and this results in the database performing a lot of extra I/O operations. This means paging I/O and database I/O have to be balanced, and a compromise has to be reached, as shown in Figure 43.

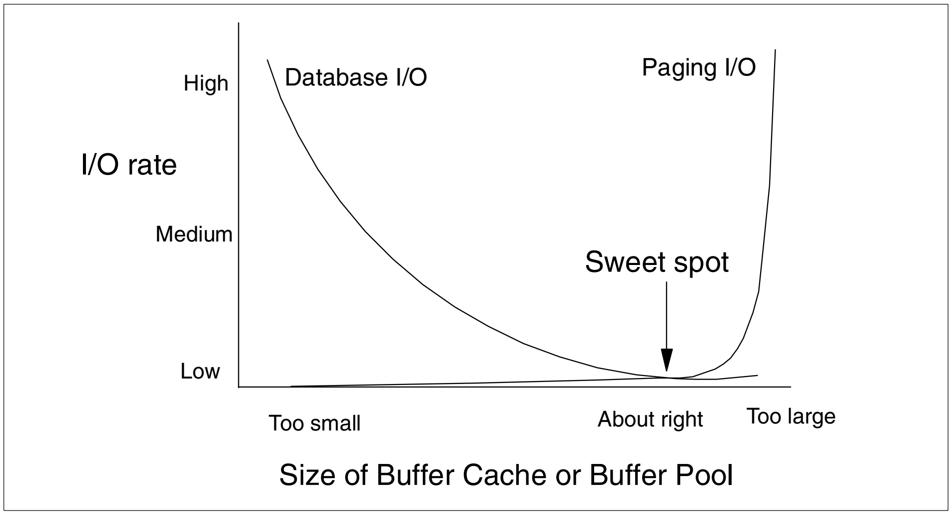

Figure 43. Balancing paging I/O against database I/O

To determine if the database buffer cache or buffer pool is of sufficient size, each database provides tools to provide details of the cache hit ratio. For OLTP systems, this is normally in the region of 95 to 99 percent. The other main usage of the memory by the RDBMS is for the shared_pool and log buffers.

16.2.4.1 Insufficient disk I/O

If the database does not have sufficient disk I/O capability, this can be clearly seen by monitoring the disk performance statistics and CPU statistics. If the machine has high I/O wait CPU numbers, this *can* indicate I/O problems, but care has to be taken in trusting this number. First, I/O wait is assigned in a manner that is not clear to many people and has been changed in AIX 4.3.3 to make more sense. Refer to Section 11.1.37, “vmstat” on page 324 for more details.

If the disk statistics are investigated, then there are three areas to check:

- Disk I/O is distributed across the disk evenly. It is the job of both the system administrator and the database administrator to ensure the I/O is spread across many disks. If one disk is overworked, and the others underused, then the whole system performance can suffer. The `iostat` command should be used to investigate this issue.
- Disks are not overworked. For a disk to perform well and respond quickly to user demands, it is best to have the disk not running above 50 percent busy, because it will mean that the queue for devices drivers can become large and, as a result, response times grow. For large query DSS and batch workload, the disks can be used above 50 percent busy. The `iostat` command should be used to investigate this issue.
- Disk channel is not a limiting factor. The disk channel is the PCI or MCA bus, SCSI, FCAL or SSA adapters, cables, and SSA loops. Any of these can become limiting factors, especially when a lot of disks are attached on the same channel. It is very hard to track if the channel is overworked and to prove it is a limiting factor. The disks will appear to not be overworked, but the CPU appears to be in I/O wait state. The `nmon` tool gives the adapter busy and throughput statistics by adding up the statistics for all the disks attached to the adapter. Also you can use SSA tools included in AIX operating system. See AIX documentation for more details

Note

The RDBMS logs (if placed on a dedicated disk or set of disks) are an exception to most of the rules above. Logs should not be spread across disks, are often overworked, and, if necessary, should have dedicated adapters and cables.

16.2.5 Insufficient network resources

The general rule of thumb is to monitor network throughput and do not allow it to be over 50 percent of the maximum throughput. Any collision detect protocol network suffers with throughput problems above this level of network traffic. See Chapter 4, “Network tuning” on page 21 for more details about this topic.

16.2.6 Insufficient logical resource access

Within AIX and the RDBMS, access to logical resources is carefully controlled to make sure that data is not corrupted. This control is implemented as locks, latches, and semaphores. But whatever method is used, this restriction means that many operations can not be performed in parallel, and for important resources, the tasks are serialized. This can have a large impact on performance.

Unfortunately, this is extremely hard to detect from observing CPU, memory, and disk activity. Only internal examination of AIX or the RDBMS will reveal the problem. AIX has trace facilities to allow this. See Section 11.1.36, “trace, trcrpt” on page 322 for more details. DB2 UDB has the snapshot facility to allow this to be investigated. See *IBM DB2 UDB Administration Guide: Performance,* SCO9-2849 for more details.

16.2.7 What can we tune?

It is easy to think the only items we can change is a few disks and the database tuning parameters. But there is quite a long list of things that can be changed. For example:

- A little purchasing power - Performance tuning should always highlight which component of the system should be considered for the next upgrade. If this is relatively inexpensive (similar to a few days performance tuning consultation), then it might be better to upgrade the machine immediately rather than continue tuning. Also, it allows planning and budgets to be prepared for longer term upgrades.
- Balancing the use of CPU, memory, and disk I/O - If one is overworked, you might be able to use the others to compensate.
- Balancing the use of memory between AIX, user processes, RDBMS processes, and the RDBMS shared memory.
- Balancing the various consumers of the RDBMS shared memory (buffer, library, locks).
- Tuning disks by balancing speed over reliability with options, such as RAID 5, striping, and mirrors.
- Changing data placement on the disks via the AIX LVM options center, middle, or edge.
- Removing hot spots by moving data between disks or hardware spreading of data via stripes or RAID 5.
- Dedicating disks to particular tasks for maximum response time and throughput. For example: the log disks.
- Ensuring equal use of disk and network I/O across adapters and that they are not approaching their theoretical or practical limits.
- Balancing disk I/O against memory. One of the many benefits of using memory is to reduce time-consuming disk I/O.

- Ensuring all CPUs of SMP machines are at work. Many performance problems are caused by a single batch process not making use of all the CPUs in the system.
- Maximizing backup rate to reduce the backup window or minimizing user interference with online backups. These considerations might affect the usage of disks and reserving disks for backup purposes.
- Balancing workloads. Many performance problems are simply poor management of workloads, in particular, batch operations or online requested reports. Sometimes users are willing to change their working habits if they realize they can improve performance (and their job) by making small changes to their work patterns.
- Identifying and fixing poor application modules and SQL statements.

  The database can help you work out the worst offenders. Many DBAs assume they cannot change the application code or SQL. This means they never investigate the SQL or try to get it improved. Having identified the worst SQL examples and the ones used repeatedly, these will yield the largest performance improvements:

  - If you know other sites that use the same application or SQL, then find out if they have the same list of issues.
  - Join the appropriate user groups.
  - Start providing feedback to the vendor or your own development team and start increasing the pressure for getting these problems fixed.
  - Although many developers resist making individual changes to application code, they do welcome real production feedback to improve their products performance in the longer term and for future releases.
- Start to think in parallel on SMP machines. Unless you can make use of all the CPUs, you will be making use of a fraction of the available compute power.
- Start to turn your attention toward logical resources, such as lock, spin counts, time-out, delay flags, and database latches, when the machine looks idle but responds badly.
- Finally, tuning the database via the parameters and options.

16.2.8 Top 20 DB2 parameters

The following topics show the most important database manager configuration (dbm) and database configuration (db) parameters. These are parameters that have a large impact on performance and should be tuned first.

To find a complete information about database manager, database and environment configuration parameters and the way to tune them, you can refer to *IBM DB2 UDB Administration Guide: Performance,* SC09-2840 or *Database Performance on AIX in DB2 and Oracle Environment,* SG24-5511.

16.2.8.1 Buffer pool size (buffpage)

The buffer pool is the area of memory where database pages (table rows or indexes) are temporarily read and manipulated. All buffer pools reside in global memory, which is available to all applications using the database. The purpose of the buffer pool is to improve database performance. Data can be accessed much faster from memory than from disk.

Note

The configuration of one or more buffer pools is the single most important tuning area because it is here that most of the data manipulations take place for applications connected to the database. This is valid for regular data only (except large objects and long field data).

Never leave this parameter on its default value.

Buffer pool performance should be tracked permanently.

- In an OLTP environment, it is recommended to allocate as much as 75 percent of the system's memory that is left after taking out the memory required by the operating system, applications running on the system, and memory for communication to the buffer pools, under the following conditions:
  - There are multiple users connected to the database.
  - This system is used as a database server only.
  - The applications access the same data and index pages repeatedly.
  - There is only one database installed.

  Particularly in OLTP environments, where there is typically a repetitive access to indexes, it is recommended to strive to have a buffer pool large enough to keep all indexes in memory.
- In a DSS environment, it is recommended to allocate up to 50 percent of the left over memory to the buffer pool. More memory is required to the database sort parameters for large queries.

The *buffer pool hit ratio* (percentage of time that the database manager did not need to load a page from disk into memory) should reach 100 percent.

This can be monitored by DB2 UDB monitor utility (Snapshot monitor or Event monitor).

16.2.8.2 Number of I/O servers (num_ioservers)

DB2 UDB can activate prefetchers that read data and index pages into the buffer pool by anticipating their need with an application (asynchronously). To enable prefetching, the database manager starts separate threads of control, known as I/O servers, to perform page reading. As a result, the query processing is divided into two parallel activities: Data processing (CPU) and data page I/O. The I/O servers wait for prefetch requests from the CPU processing activity.

Because one I/O server can serve only one I/O device (disk), it is recommended to configure one or two more num_ioservers than the number of physical devices on which the tablespace containers reside. It is better to use additional I/O servers because there is a minimal overhead associated with each.

16.2.8.3 Number of asynchronous page cleaners (num_iocleaners)

Page cleaners are DB2 UDB processes that monitor the buffer pool and asynchronously write pages to disk before the space in the buffer pool is required by another database agent. This means that the agents will not wait for changed pages to be written out before being able to read a page.

- In an OLTP environment, where many transactions are run against the database, it is recommended to set the value of this parameter to between one and the number of physical storage devices used for the database. Environments with high update transaction rates may require more page cleaners to be configured. This is also valid for database systems with large buffer pools.
- In a DSS environment that will not have updates, it is usual to set this parameter to 0. The exception would be if the query workload results in many TEMP tables being created. This can be determined by using the Explain utility. In this case, it is recommended to set the number of I/O cleaners to the number of disks that assigned to the TEMP tablespace.

You can use the command

```
> get snapshot for bufferpools on database_name
```

to monitor the write activity information from the buffer pools in order to determine if the number of page cleaners must be increased or decreased.

16.2.8.4 Changed pages threshold (chngpgs_thresh)

This parameter can be used to specify the level (percentage) of changed pages at which the asynchronous page cleaners will be started if they are not currently active. When the page cleaners are started, they will build a list of the pages to write to disk. Once they have completed writing those pages to disk, they will become inactive again and wait for the next trigger to start. Therefore, this parameter is connected to the num_iocleaners parameter.

- In an OLTP environment, you should generally ensure that there are enough clean pages in the buffer pool by setting the chngpgs_thresh value to be equal to or less than the default value. A percentage larger than the default (60 percent) can help performance if the database has a small number of very large tables.
- In an DSS environment, these page cleaners are not used.

16.2.8.5 Sort heap size (sortheap)

The sortheap is a database configuration parameter. It is the amount of private memory allocated to each process connected to a database at the time of the sort. The memory is allocated only at the time of sort and deallocated after sorting has been finished. It is possible for a single application to have concurrent sorts active. The larger the table to be sorted, the higher the value should be for this parameter. If the value is too large, then the system can force to page if memory becomes overcommitted.

This is one of the most important areas to be tuned because a sort operation done in real memory can significantly improve performance. It is recommended that the remaining real memory that is not allocated to the AIX, applications, and other DB2 UDB memory structures is allocated to the sort operations.

If there is more data to be sorted than memory space, merge phases will be required in order to finish the sort operation. A possible way to avoid this is to increase the sortheap parameter.

The `get snapshot for database database_name` command will provide two indicators that can be used for tuning the sortheap parameter:

- Total sort time
- Total sorts

It is recommended that you keep on increasing the sortheap parameter as long as both of the following conditions are true:

- You have real memory available.
- The result value of the equation *total sort time/total sorts* is decreasing.

16.2.8.6 Sort heap threshold (sheapthres)

This is a database manager configuration parameter. DB2 UDB uses this parameter to control the sum of all sortheap allocations of all applications in the instance. Therefore, this parameter impacts the total amount of memory that can be allocated across the database manager instance for sortheap.

Explicit definition of the threshold prevents the database manager from using excessive amounts of memory for large numbers of sorts. It is recommended to set this value to a reasonable multiple of the largest sortheap parameter defined in the database manager instance. This parameter should be at least two times the largest sortheap value for any database within the instance.

It is important to be aware that, when using DB2 UDB V5.2, and when the database manager parameter intra_parallel is enabled, the sheapthres does not simply work as a limiting number anymore, but the amount of space defined for this parameter is automatically allocated from memory.

16.2.8.7 Statement heap size (stmtheap)

The statement heap size is a database configuration parameter that specifies the size of workspace used for the SQL compiler during the compilation of an SQL statement. For dynamic SQL statements, this memory area will be used during execution of the application. For static SQL statements, it is used during the bind process. The memory will be allocated and released for every SQL statement only.

For most cases, the default value, 204 pages, can be used. If an application has very large SQL statements, and DB2 UDB reports an error when it attempts to compile a statement, then the value of this parameter has to be increased. The error messages issued are:

- `SQL0101N The statement is too long`
- `SQL0437W Performance of this complex query may be sup-optimal. Reason code 1.`

These messages are sent to the applications that run the queries and are also logged in the DB2 UDB error log file called *db2diag.log*.

16.2.8.8 Package cache size (pckcachesz)

This database configuration parameter is used to define the amount of memory for caching static and dynamic SQL statements. Caching packages allows the database manager to reduce its internal overhead by eliminating the need to access the system catalogs when reloading a package or, in the case of dynamic SQL, eliminating the need for compiling a query twice.

The package cache is important in an OLTP environment where the same query is used multiple times by multiple users within an application. To tune this parameter, it is helpful to monitor the package cache hit ratio. This value shows if the package cache is used effectively. If the hit ratio is large (> 90 percent), the package cache is performing well.

The package cache hit ratio can be obtained by the following formula:

(1 - (package cache inserts / package cache lookups)) * 100 percent

These indicators can be retrieved by the `get snapshot for database on database_name` command.

16.2.8.9 Database heap size (dbheap)

Each database has one memory area called a database heap. It contains control block information for tables, indexes, table spaces, and buffer pools. It also contains space for the event monitor buffers, the log buffer (logbufsz), and the catalog cache (catalogcache_sz). The memory will be allocated when the first application connects to the database and keeps all control block information until all applications are disconnected.

Each page in the buffer pool has a descriptor of about 140 bytes. For every 30 buffer pool pages, an additional page for overhead is needed in the database heap. For databases with a large amount of buffer pool, it is necessary to increase the database heap appropriately.

16.2.8.10 Maximum number of active applications (maxappls)

This database parameter specifies the maximum number of concurrent applications that can be connected (both local and remote) to a database. Since each application that attaches to a database causes some private memory to be allocated, allowing a larger number of concurrent applications will potentially use more memory.

Increasing the value of this parameter without decreasing the maxlocks parameter or increasing the locklist parameter can cause the database's limit on locks to be reached more frequently, resulting in many lock escalation problems.

16.2.8.11 Maximum number of agents (maxagents)

This parameter indicates the maximum number of database manager agents (db2agent) available at any given time to accept application requests.There are two types of connections possible that require the use of DB2 UDB agents. Local connected applications require db2agents within the database manager instance as well applications running on remote clients. The maxagents parameter value must be at least equal to the sum of both values. This parameter is useful in memory constrained environments to limit the total memory usage of the database manager because each additional agent requires additional memory.

The value of maxagents should be at least the sum of the values for maxappls in each database allowed to be accessed concurrently.

16.2.8.12 Maximum storage for lock list (locklist)

This parameter indicates the amount of storage that is allocated to the lock list. There is one lock list per database, and it contains the locks held by all applications concurrently connected to the database. Locking is required to ensure data integrity; however, too much locking reduces concurrency. Both rows and tables can be locked.

If the memory assigned for this parameter becomes full, the database manager performs lock escalation. Lock escalation is the process of replacing row locks with table locks, reducing the number of locks in the list. DB2 UDB selects the transaction using the largest amount of the locklist and changes the record locks on the same table to a table lock. Therefore, one lock (table lock) replaces many locks (record locks) of a table. This reduces the concurrency and the performance.

If lock escalations are causing performance or hang problems, it is recommended to increase the value of this parameter.

16.2.8.13 DB2MEMDISCLAIM and DB2MEMMAXFREE

Depending on the workload being executed and the pool agents configuration, it is possible to run into a situation where the committed memory for each DB2 UDB agent will stay above 32 MB even when the agent is idle. This behavior is expected and usually results in good performance, as the memory is available for fast reuse. However, on a memory constrained system, this may not be a desirable side effect. The DB2 command

```
> db2set DB2MEMDISCLAIM = yes
```

avoids this condition. This variable tells the AIX operating system to stop paging the area of memory so that it no longer occupies any real storage. This variable tells DB2 UDB to disclaim some or all memory once freed depending on DB2MEMMAXFREE. This ensures that the memory is made readily available for other processes as soon as it is freed. DB2MEMMAXFREE specifies the amount of free memory that is retained by each DB2 UDB agent. It is recommended to set this value to 8 MB by using:

```
> db2set DB2MEMMAXFREE = 8000000
```

16.2.8.14 DB2_PARALLEL_IO

This registry variable can be used to force parallel I/O for a tablespace that has a single container. When reading data from, or writing data to, tablespace containers, the database manager may use parallel I/O if the number of containers in the database is greater than 1. However, there are situations when it would be beneficial to have parallel I/O enabled for single container tablespaces. For example, if the container is created on a single RAID device that is composed of more than one physical disk, you may want to issue parallel read and write calls. The DB2_PARALLEL_IO variable can be set to one specific tablespace or to all tablespaces of that instance. For example, to enable parallel I/O for tablespace USERSPACE1 with tablespace ID 2, the command is:

```
> db2set DB2_PARALLEL_IO = 2
```

16.2.8.15 DB2_STRIPED_CONTAINERS

When creating a DMS tablespace container (device or file), a one-page tag is stored at the beginning of the container. The remaining pages are available for data storage by DB2 UDB and are grouped into extent-sized blocks. When using RAID devices for tablespace containers, it is suggested that the tablespace is created with an extent size that is equal to, or a multiple of, the RAID stripe size. However, because of the one page container tag, the extents will not line up with the RAID stripes, and it may be necessary during an I/O request to access more physical disks than would be optimal. DMS table space containers can now be created in such a way that the tag exists in its own (full) extent. This avoids the problem described above, but it requires an extra extent of overhead within the container. To create containers in this fashion, set the DB2 UDB command:

```
> db2set DB2_STRIPED_CONTAINERS=ON
```

Any DMS container that is created will have new containers with tags taking up a full extent. Existing containers will remain unchanged.

16.2.8.16 Tablespace page size

Since Version 5.2, DB2 UDB provides the possibility to expand the page size of tablespaces from the default 4 KB to 8 KB. Version 6.1 supports 16 KB or 32 KB. This allows larger tables and larger row lengths. For example, a table created in a tablespace with a page size of 32 KB can reach a maximum size of 512 GB.

A bufferpool using the same page size must be assigned to each tablespace. For instance, if you have tablespaces with 4 KB, 8 KB, and 16 KB page sizes within a database, the database must have at least three bufferpools that also use a page size of 4 KB, 8 KB, and 16 KB.

A tablespace page size larger than 4 KB is advantageous for tables with large data rows. However, it is important to be aware that on a single data page there will never exist more than 255 rows of data, regardless of the page size.

16.2.8.17 REORG

The performance of SQL statements that use indexes can be impaired after many updates, deletes, or inserts have been made. Newly inserted rows can often not be placed in a physical sequence that is the same as the logical sequence defined by the index (unless you use clustered indexes). This means that the Database Manager must perform additional read operations to access the data because logically sequential data may be on different physical data pages that are not sequential. The DB2 UDB `REORG` command performs a reorganization of a table by reconstructing the rows to eliminate fragmented data and by compacting information.

16.2.8.18 REORGCHK

Because the REORG utility needs a lot of time for reorganizing a table, it is useful to run the `REORGCHK` command before running the `REORG` command. The REORGCHK utility calculates statistics on the database to determine if tables need to be reorganized or not.

16.3 DB2 UDB Implementation on RS/6000 SP: a real case

This section describes the latest performance record reached by IBM in TPC-H, working with DB2 UDB 7.1 on RS/6000 SP platform, with AIX 4.3.3. We detail here the complete infrastructure (hardware and software) used to get it and the way this environment was configured and tuned.

You can find some relevant information in this report, but do not forget it is only one case of DSS workload. Do not take it as an IBM's recommendation or general rule.

16.3.1 Hardware configuration

The RS/6000 SP configuration is:

- 32 wide nodes: 4-way SMP with 375 MHz Power3-II processors
- 4 GB memory per node
- 4 internal 9.1 GB SCSI disks per node
- 2 SSA adapters per node
- 4 frames
- 2 SP switch, each witch 4.8 GB/sec peak

- Control Workstation: RS/6000 model F50
- SSA disks:
  - (11) 7015 Expansion Racks
  - (64) 7133-D40 disk drawers, each one with (16) 9.1 GB drives (10,000RPM)
  - 9.32 TB disk in 992 drives + 32 hot spares

For this measurement, wide nodes were used in order to exploit the additional I/O bandwidth. To determine the number of nodes, the typical rule of thumb was used; between 60 GB and 200 GB of raw data per node, although many factors come into play when deciding this topic. For example, a high percentage of unqueried data allows more data per node, while a low percentage suggests less data per node.

Each node has an identical disk layout as shown in Figure 44 on page 422. All data objects were spread across a total of 31 disks, 16 on the first adapter and 15 on the second, with one disk reserved as a hot spare. Experience has shown that reserving one disk for this purpose is well worth the slight increase in expense and loss of performance, because the disk can be swapped in to replace a failed disk very quickly.

This configuration results in a good balance of disk-to-adapter bandwidth, with each adapter running very near its limit of about 90MB/sec. Only two of the three available PCI buses in the nodes were used, however.

Note

Ideally, a minimum of eight to sixteen disks per CPU is needed for optimal performance in a balanced system.

In this benchmark, eight disks per CPU were configured.

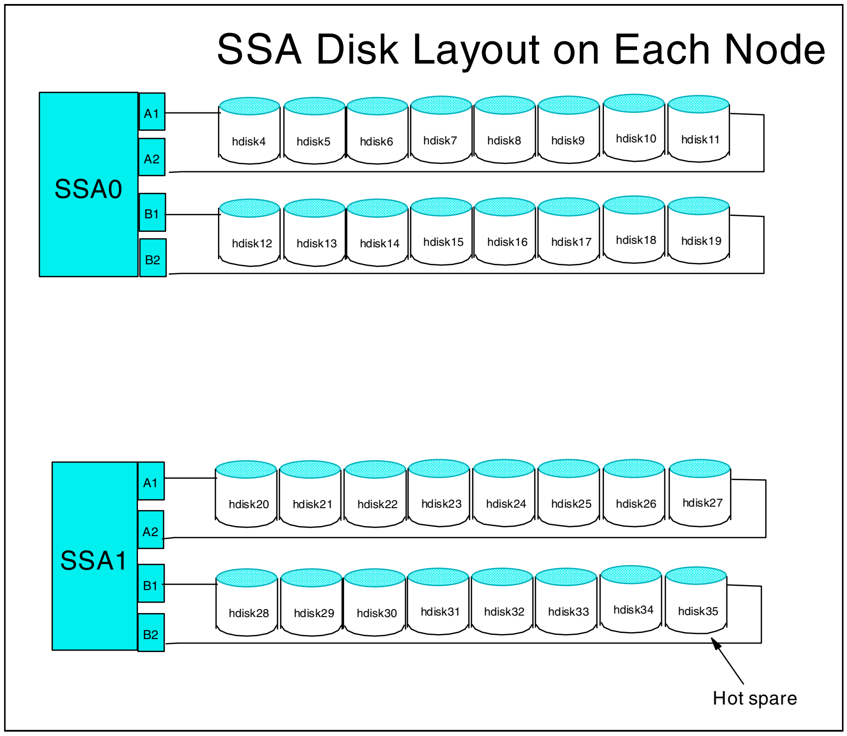

Figure 44. The SSA disk layout.

16.3.2 Software configuration

For ease of management, all nodes have the same software at the same levels:

- AIX 4.3.3
- PSSP 3.1.1
- DB2 UDB EEE V7.1
- IBM C Compiler V4.4
- Performance Toolbox V2.2

16.3.3 Database layout

The goal was to press the maximum number of disks into service, while attempting to find some methods of reducing disk head movement.

Of the 32 disks per node, one was reserved as a hot spare and the remaining 31 disks were used for containers for all tablespaces. As indicated in Figure 45, data objects which are frequently accessed concurrently were placed in adjacent physical partitions. Also, all primary logical volume copies were grouped together in the center of each disk and placed in specific disk partitions using the commands:

```
# mklv -m <mapfile>
# mklvcopy -m <mapfile>
```

The *parallel-sequential* scheduling policy, available in AIX 4.3.3, was set using the flag `-d ps` on mklv (chlv). This caused the primary copy of each logical volume to always be used for reads, except in the case of disk failure, and writes to those logical volume copies are done in parallel. We also gained some performance efficiency by turning off mirror write consistency:

```
# chlv -wn <lvname>
```

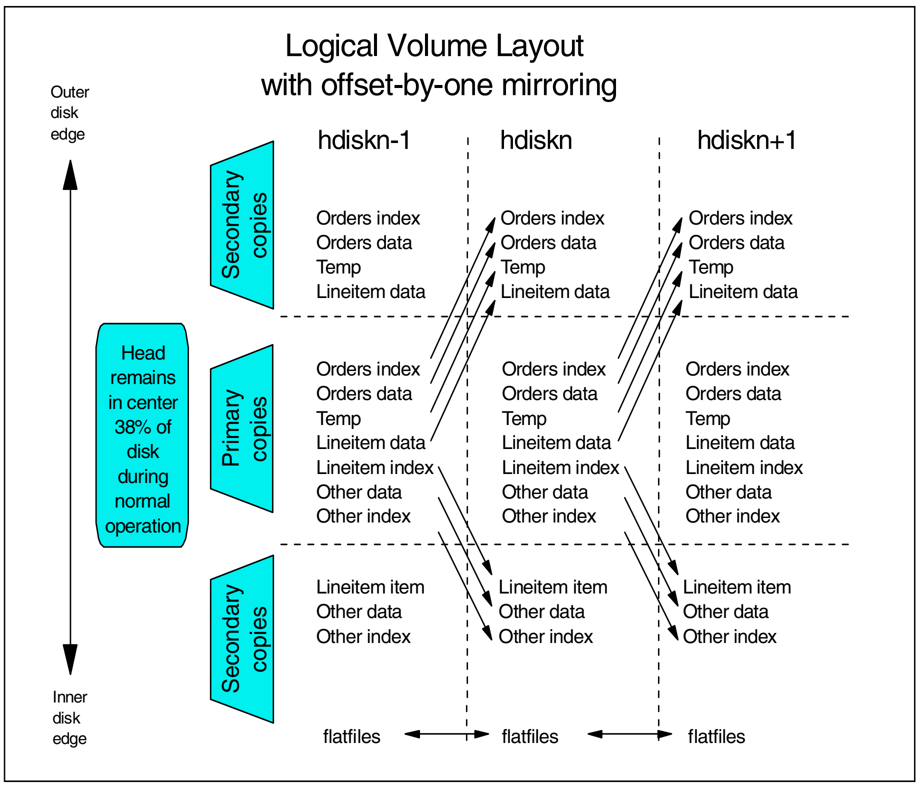

Figure 45. Logical volume layout.

16.3.4 Operating system tuning

The benchmark team follows well established guidelines for commercial applications (see Section 4.7.3, “Tuning for commercial and database environments” on page 89) prior to any fine tuning of performance once the tuning of the database started. Some of the basic guidelines, as shown in Table 43, include:

- Paging space and memory: one-to-one relationship between paging space and memory. No paging appeared during benchmark in the power test and minimal paging during the throughput test.
- Switch and Network: switch send and receive pools set to the recommended value of 16 MB. The CWS was configured as a NFS file server for the user’s home directories. NFS socket buffers size adjusted accordingly.
- Virtual Memory Manager (VMM): of the `vmtune` parameters, maxpgahead option (-R 128) is the most important. This affects the efficiency of reading from the system temporary tablespace, which uses JFS file containers. The settings of minperm, maxperm, minfree and maxfree are intended to limit the amount of memory allocated for file buffering, in order to make more memory available to DB2.

Table 43. System parameters tuned

Paging	**no parameters**
4 GB paging space per node (1:1) Database tuned for minimal paging	thewall = 1,048,576 sb_max = 3,000,000 tcp_sendspace = 221,184 tcp_recvspace = 221,184 udp_sendspace =65,536 udp_recvspace = 655,360 rfc1323 = 1
/usr/samples/kernel/vmtune	**nfso parameters**
-R (maxpgahead) = 128 -f (minfree) = 128 -F (maxfree) = 256 -c (numclust) = 1 -k (npskill) = 512 -w (npswarn) = 2000 -p (minperm) = 5 -P (maxperm) = 15	nfs_socketsize = 2,000,000 nfs_tcp_socketsize = 442,240
maxuprocs	**switch parameters**
4000	spoolsize = 16,777,216 rpoolsize = 16,777,216

Go to IBM Business Intelligent web page (`http://www.rs6000.ibm.com/solutions/bi`) or in TPC Organization web page (`http://www.tpc.org`) for a complete list of all AIX parameters.

16.3.5 Database tuning

This section lists some DB2 parameters tuned in the benchmark, as shown in Table 44. Again, for a more detailed list, go to IBM DB2 web page or TPC Organization web page. Also, you can find a complete table with the most important configurable database parameters and their impact on performance in the redbook *Database Performance on AIX in DB2 UDB and Oracle Environments,* SG24-5511.

Table 44. DB2 UDB parameters tuned

Exploit hardware	**Optimize query plans**
cpuspeed comm_bandwidth intra_parallel max_querydegree dft_degree	db2_like_varchar db2_sort_after_tq db2_correlated_predicates transferrate db2_vector db2_pred_factorize db2_antijoin
Adjust memory-related parameters	**Optimize database runtime efficiency**
buffpage sortheap sheapthres db2memmaxfree locklist maxlocks db2_memdisclaim	prefetchsize num_ioservers num_iocleaners db2_mmap_write db2_mmap_read db2_rr_to_rs

16.3.6 System management and performance

The key to managing systems of this size is to seize every opportunity to automate, document, and collect relevant data in order to minimize problems and avoid repetitive and tedious tasks. Some of the things done in our system that you can use are:

1. Use the inherent parallel structure of the RS/6000 SP and PSSP code by rebooting nodes, doing system maintenance in parallel, using the `dsh` command to propagate commands to all nodes simultaneously, and so on. We kept all nodes as identical as possible, including the I/O subsystem. Many of the database functions also take advantage of this symmetry.
2. Write generic scripts and maintain tools of common tasks, log all activities and users on the system, create checklists of items to verify following a reboot, prior to starting the database, and so on.
3. Certify disks at regular intervals to detect failing disks early and develop an effective disk replacement procedure with the use of hot spares.

4. Monitor error logs daily, and have the system automatically send a page to the systems administrator if specific events occur.
5. Collect detailed performance information on key metrics with the use of:
   - *Performance Toolbox*: The nodes are monitored using the `ptxrlog` utility using the variables listed in Table 45. The real time performance monitor *3dmon* was used to look at system resource imbalances.

Table 45. Variables used by ptxrlog

Variable	Comment
CPU/cpu0/kern	Kernel CPU utilization percent
CPU/cpu0/user	User CPU utilization percent
CPU/cpu0/wait	I/O wait percent
Mem/Virt/pgspgin	Physical page-ins from paging space
Mem/Virt/pgspgout	Physical page-outs to paging space
Mem/Real/numfrb	Number of memory pages on the free list
IP/NetIF/en0/ioctet	Bytes received over the ethernet
IP/NetIF/en0/ooctet	Bytes transmitted over the ethernet
IP/NetIF/css0/ioctet	Bytes received over the switch
IP/NetIF/css0/ooctet	Bytes transmitted over the switch
Disk/hdisk0/busy	hdisk0 (internal SCSI) percent busy
Disk/hdisk1/busy	hdisk1 (internal SCSI) percent busy
Disk/hdisk4/busy	hdisk4 (external SSA) percent busy
Disk/hdisk14/busy	hdisk14 (external SSA) percent busy

   - `iostat`: Used to ensure even utilization across disks and monitor aggregate I/O rates.
   - `vmstat`: Used to monitor CPU utilization and paging activity.
   - `netstat`: Used to monitor switch and ethernet traffic.

The following pictures are some of the graphics obtained from the benchmark. The purpose of the charts is not to document a detailed analysis but to illustrate typical behavior of the system during query and update activity, and difference between power and throughput runs.

16.3.6.1 Power run

The CPU activity for the power run shows a fair amount of variation between user busy and I/O wait time due to the inherent nature of the queries (as shown inFigure 46); some have higher I/O requirements than others, while others are CPU intensive. During the entire power run, kernel busy time remained much lower than the other two metrics.

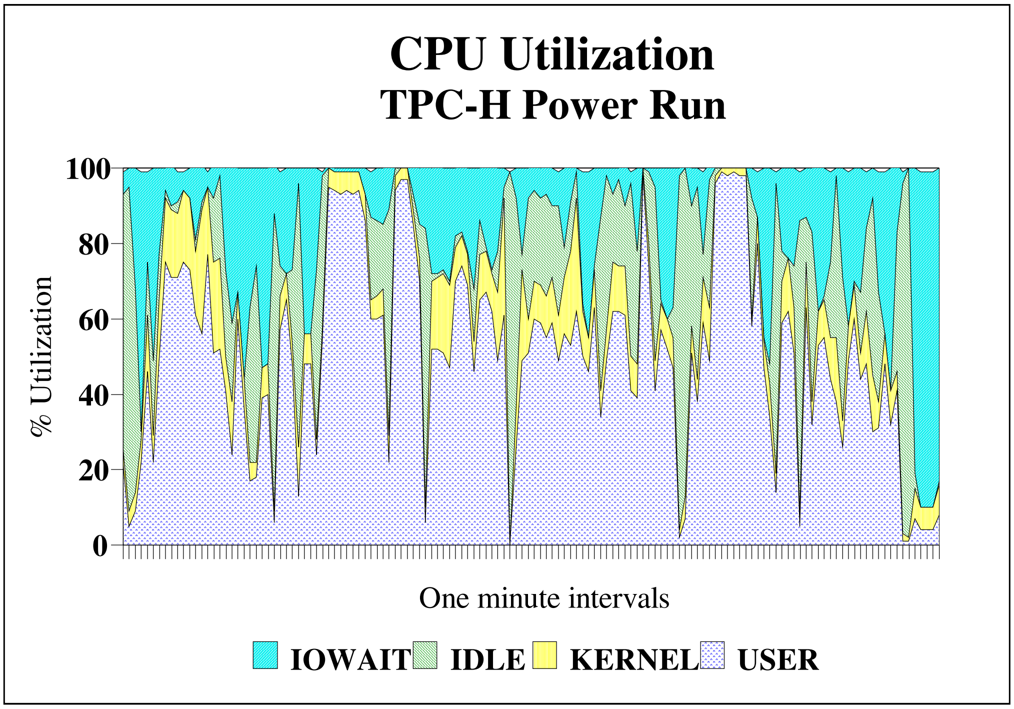

Figure 46. CPU utilization. TPC-H power run.

The activity on the SSA disks is typical of all database disks. Some paging activity was present only during updates (beginning and end of the chart) and in a couple of queries. It can be observed that I/O requirements vary greatly by query ranging from an aggregate MB/s per node from none to peaking at 175 MB/s and averaging 82 MB/s for the entire power test.

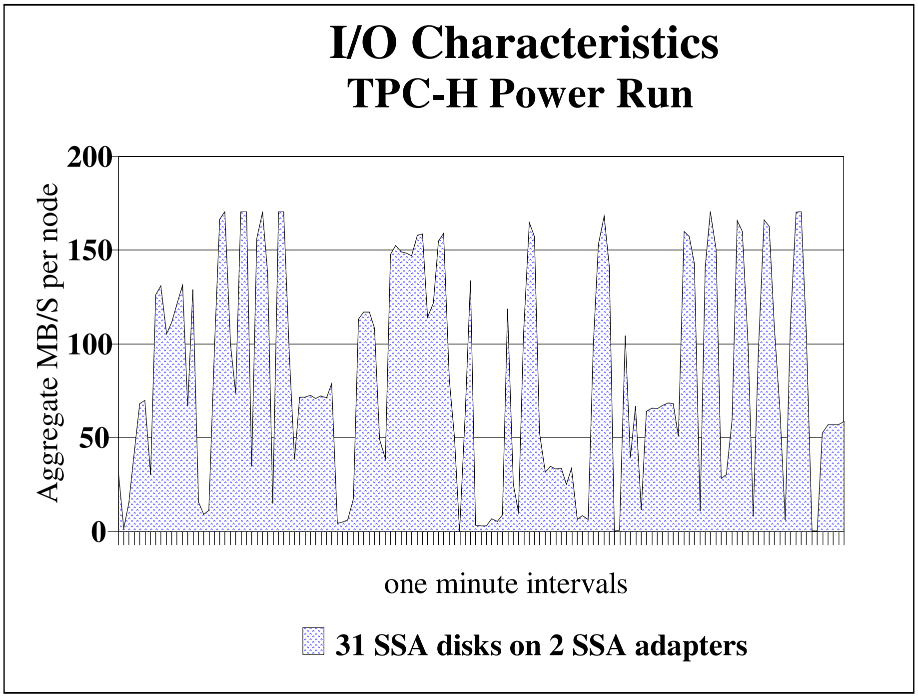

Figure 47. I/O characteristics. TPC-H power run

16.3.6.2 Throughput run

In the case of the throughput run, seven concurrent queries are executing at any given time, followed by updates; these two phases (queries + updates) are evident in the CPU utilization chart. During the queries, CPU and kernel busy times are dominant, and there is small amount of I/O wait, which indicates that we are still I/O constrained at times. Having more than one stream of queries running concurrently allows for full use of the processors. During the update phase, I/O wait goes up. (Refer to Figure 47 for an overview.)

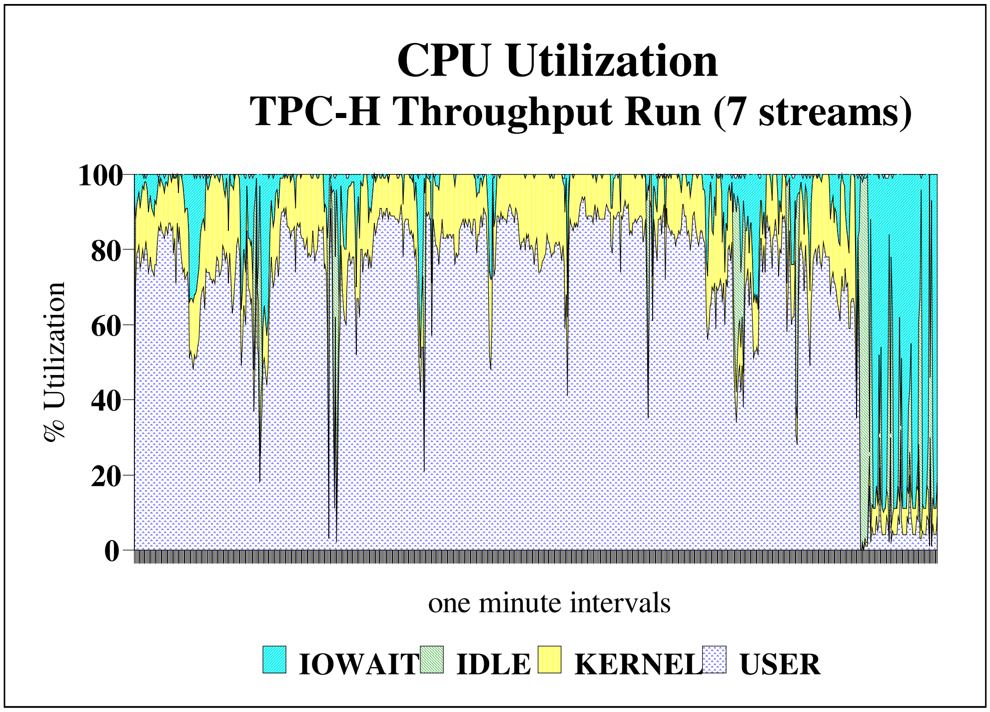

Figure 48. CPU utilization. TPC-H throughput run.

The disks' throughput shows a different pattern towards the end the run indicating that the queries are completed and the updates are running, as shown in Figure 48. The aggregate I/O bandwidth during the throughput measurement, as shown in Figure 49 on page 430, is more consistently close to its average 114 MB/s, 30% higher than the power run aggregate.

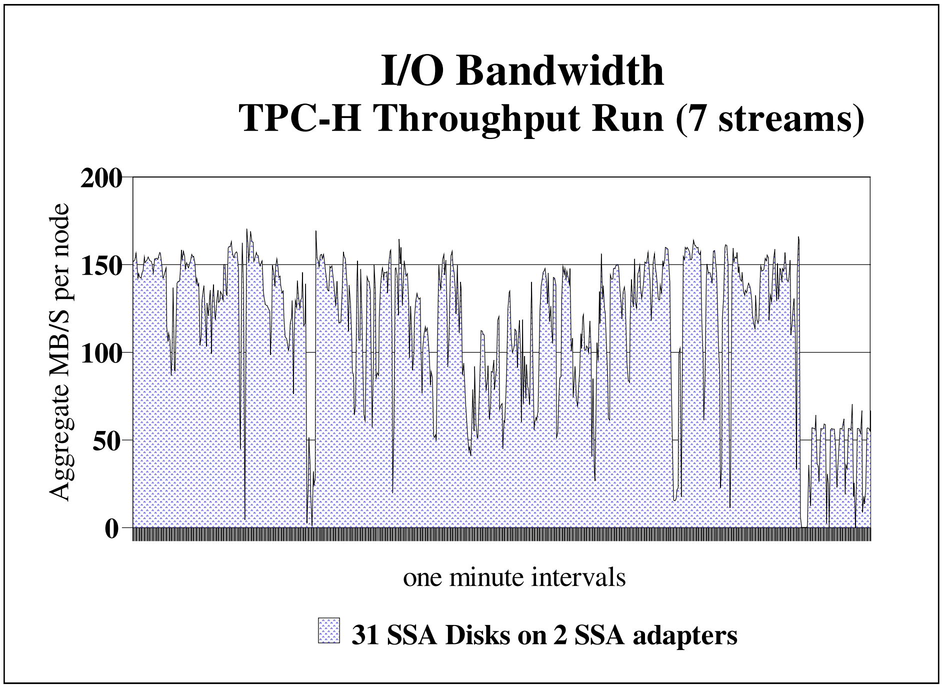

Figure 49. I/O bandwidth. TPC-H throughput run.

16.3.7 Additional tuning observations

These are some tuning observations extracted from the test:

- Physical database layout is a very important factor
  - Data distribution on disks (data, index, temp)
  - Minimize head movement, maximize arms
  - Minimal change observed with *SSA fast write cache* on
  - Experiment with pagesizes, prefetchsizes for 128K I/O
- Memory management is a balancing act
  - Multiple bufferpools provide flexibility
  - Large sortheap space needed for multi-stream work
- Change Decimal datatype to Float datatype has significant impact
- Monitoring overhead is noticeable
- Excellent *intra-parallel* speedups

- I/O bound queries don't benefit

- Use row locking to improve performance

Chapter 17. Oracle

The purpose of this chapter is to provide information about the parallel version of Oracle and allow you to gain an understanding of how the parallel version work, how it benefits from the RS/6000 SP architecture, and in which workload environments it is most efficiently used.

The Oracle8*i* Parallel Server (OPS) is a resource-sharing system that increases availability and performance by partitioning the workload across multiple nodes.

For a detailed information about AIX go to these manuals:

- Understanding IBM RS/6000 Performance and Sizing, SG24-4810

Also, check these IBM redbooks:

- *Oracle8i Parallel Server on IBM SP Systems: Implementation Guide*, SG24-5591
- *Database Performance on AIX in DB2 UDB and Oracle Environments*, SG24-5511
- *RS/6000 Performance Tools in Focus*, SG24-4989

17.1 Oracle Parallel Server (OPS)

Oracle has implemented its RDBMS product on the IBM RS/6000 SP so that it can run on multiple nodes of an SP system in parallel. This implementation uses the *shared disk* model (every CPU has its own dedicated memory, but all processors share the same disks within the system) and is the only one in this topic, as the other parallel databases use shared nothing. The parallel version of Oracle is like classic Oracle with a few additions.

- It makes use of the Oracle Parallel Query (OPQ) features that are also available on classic Oracle, but here, not only does the query get split out onto the CPUs of a single system, but also split out across the nodes of the SP system. This feature is used a lot in DSS workloads where all the CPUs in all of the different nodes can participate in working on the query for maximum parallelization and reduced response times.
- The addition of the Oracle Parallel Server (OPS), which allows different instances of Oracle to run on different nodes of the SP system having shared access to a single database. OPS uses two extra subsystems to achieve this; the Virtual Shared Disk (VSD) and the Distributed Lock

Manager (DLM). These two components are described later in this chapter.

- Many Oracle tools have been enhanced to allow extra parallelization. For example, parallel load and parallel indexes.
- To support very large databases, the new partitioned tables features is very important. It reduces DBA time for load, index, and delete operations of very large tables, and can also reduce query time.

Many of these extra features are also available in the classic version of Oracle, but they become very important for OPS and with extremely large databases.

17.1.1 Parallel Oracle architecture

Figure 50 on page 435 shows the various components of classic Oracle. There are two major components:

1. The Instance - The processes connected and co-operating via the SGA. The *front end* processes (also referred to as client processes and server or shadow processes) are connected to users and execute the SQL statements. The *back end* processes (also referred to as background processes) are internal to Oracle for the redo log, database updating, and recovery. They come and go with an Oracle Instance.
2. The Database - All the disks and files that make up the database.

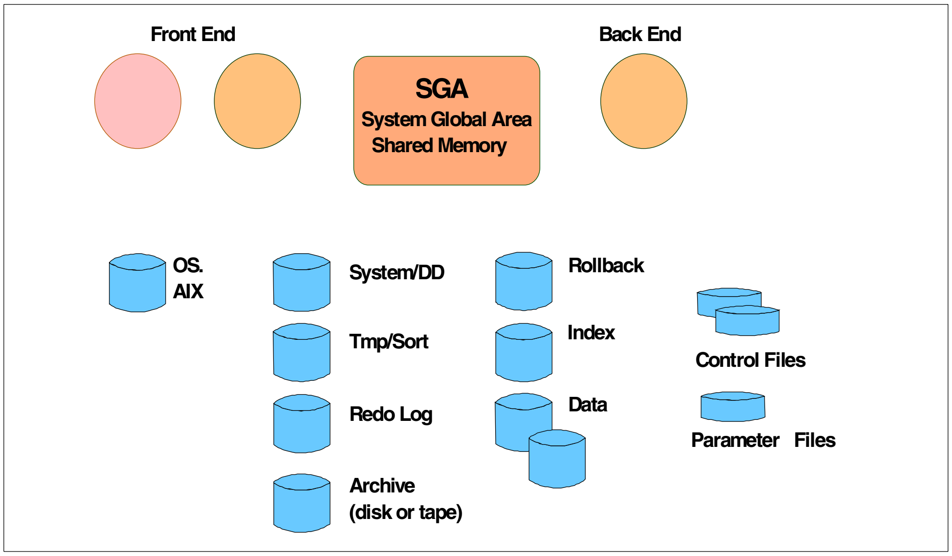

Figure 50. An Oracle instance

In a Oracle Parallel Server, there are multiple copies of the instance and a single database. Figure 51 on page 436 shows the overview level of Oracle with multiple instances and one database. The database is, of course, made up of lots of files and/or devices that are ultimately on a disk in the system. This general architecture of the OPS is then mapped to the SP architecture where each SP node is an RS/6000 computer on its own. Each node has:

- One CPU or multiple CPUs (in an SMP node)
- Memory
- Adapters
- Disks

On the SP system, there is also the High Speed Switch network (SP Switch) that allows fast communication (low latency) and high bandwidth (large data transfers) between the nodes of the SP.

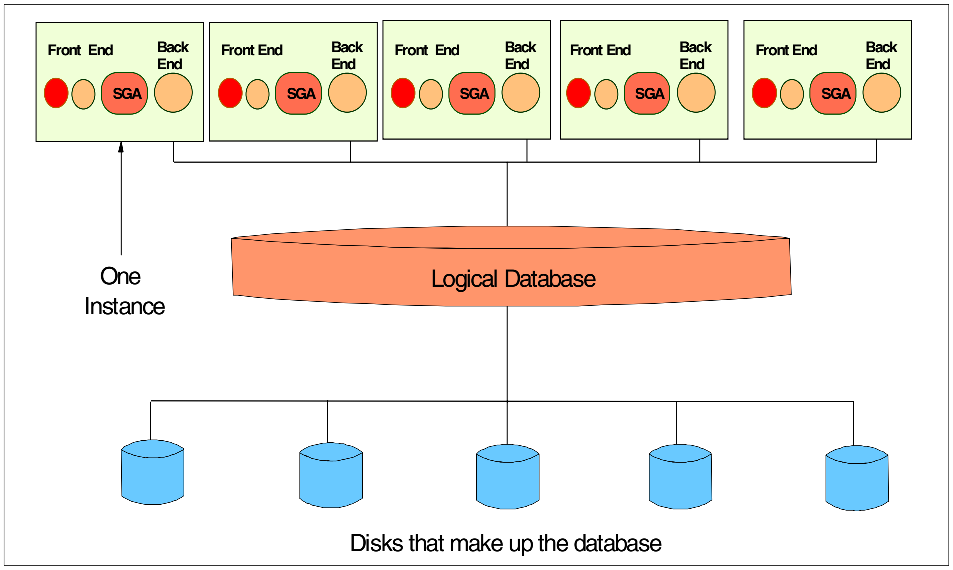

Figure 51. General Oracle Parallel Server architecture

With OPS, one instance of Oracle is run on each node of the SP, but there are two problems with this architecture:

1. An individual disk is actually attached to one of the nodes. This makes it impossible for an instance on one node to read data from disks that are attached to another node.
2. All of these instances of Oracle may have local copies of data blocks, and there is a risk of two of them updating the same data and corrupting the database.

These two problems are addressed by two software systems:

1. The *Virtual Shared Disk* (VSD) makes it possible for OPS to read the data from a remotely attached disk. The details are in Section 17.1.2, “Virtual Shared Disk (VSD)” on page 438.
2. The *Distributed Lock Manager* (DLM) makes sure data corruption between instances does not happen.The details are in Section 17.1.3, “Distributed lock manager (DLM)” on page 439.

Figure 52 on page 437 shows the architecture of OPS when implemented on the SP system. The database is spread across disks that are attached to all the nodes of the SP. This spreads the I/O requirements evenly across disks, adapters, and all of the nodes in order to reduce I/O bottlenecks.

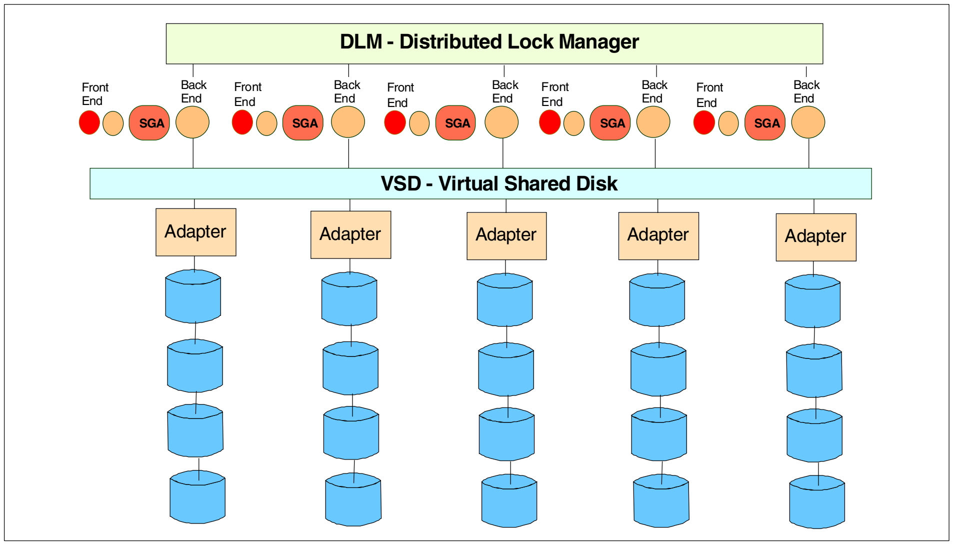

Figure 52. Parallel Oracle on SP systems

When an application connects to OPS, it actually connects to one of the instances. As every instance has access to the entire database, simple (OLTP type) queries are performed directly by that instance. If the query is large, Oracle is able to split the query into sub-queries. For example, a full table scan can be broken down into a number of sub-queries that each work on a range of rows, and other sub-queries can merge the results together and sort the results. On an SMP node, these sub-queries can be run on different CPUs. For very large queries, the sub-queries can also be sent to other nodes of the SP system, and the results are sent back to the original node. So, in Parallel Oracle, there are two types of parallelization:

- Degree - The number of processes (and, therefore, CPUs) on an instance
- Instance - The number of instances to use

The decision on the amount of parallelization is made by the optimizer and is based on the parameters set on one of the following:

- Table level parallelization settings
- Hints in the SQL statement

In general all the nodes in an OPS system on the SP are kept identical to reduce system administration and DBA workloads. Due to the extra dimension of multiple nodes and the large size of these databases, OPS is considered complex when compared to a single SMP machine running on smaller databases with classic Oracle. However, the power and scalability of the SP system does mean that much larger databases can be implemented, and this is a requirement, particularly for DSS workloads.

17.1.2 Virtual Shared Disk (VSD)

This is an IBM supplied product that allows any of the nodes in the SP system to read and write data from disks attached to other nodes. Figure 52 on page 437 shows that the VSD layer is between the Oracle Instance and the device drivers and disk adapters. OPS does not have direct access to the disks but opens VSD devices instead. The VSD layer then checks if the real disk is attached to the local node, and if not, it works out which other node has it attached:

- In the case of a local disk, the read/write request is passed by the VSD directly to the regular logical volume device driver on the node.
- In the case of a remote disk, the request is sent by the VSD device driver (via the SP Switch) to the VSD device driver on the node to which the disk is attached. The receiving VSD device driver then passes the request to the device driver of the second node, and the data is read or written on behalf of the initial node. The data is also transferred over the SP Switch.

The difference in time between the local and remote disk I/O is small.

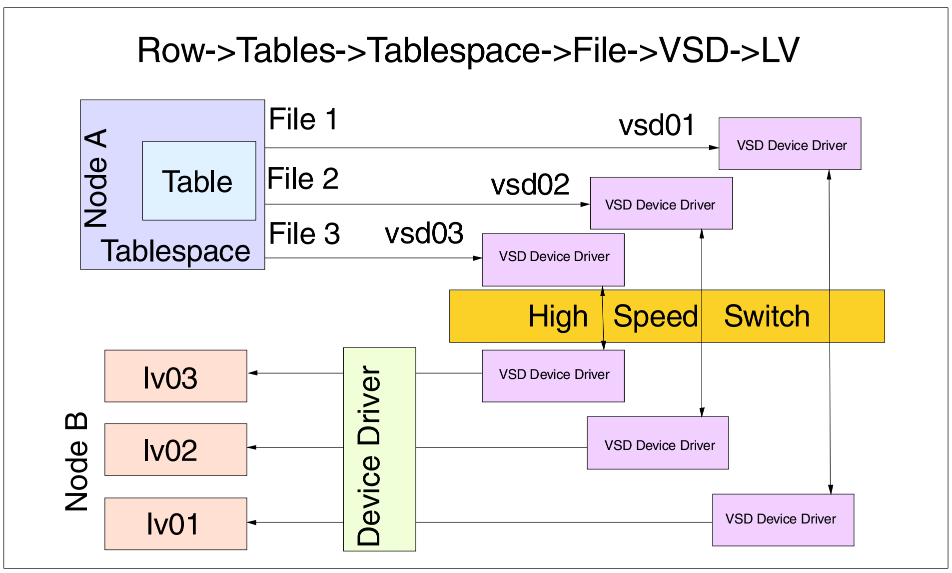

Figure 53. Oracle tables working with VSD operations

Figure 53 on page 438 shows the VSD structure in more detail. Node A thinks it has three files that make up a tablespace, but the files are actually VSDs. When Oracle reads from or writes to these files, the requests are passed over the SP Switch (high speed Switch) to the right node.

In OPS, all Oracle data files are raw devices; there are no JFS based data files except for the init.ora parameter files. Reading from, and writing to, a table means I/O operations are performed to one or more of the files that make up the tablespace in which the table resides. As far as Oracle knows, these files behave like real devices but are actually VSD devices. If necessary, the disk I/O requests are redirected to the node with the logical volumes that contain the actual data where the regular logical volume manager device driver actually does the disk I/O.

For more information about VSD tuning, refer to Chapter 13, “VSD” on page 353.

17.1.3 Distributed lock manager (DLM)

Figure 54 illustrates how DLM works:

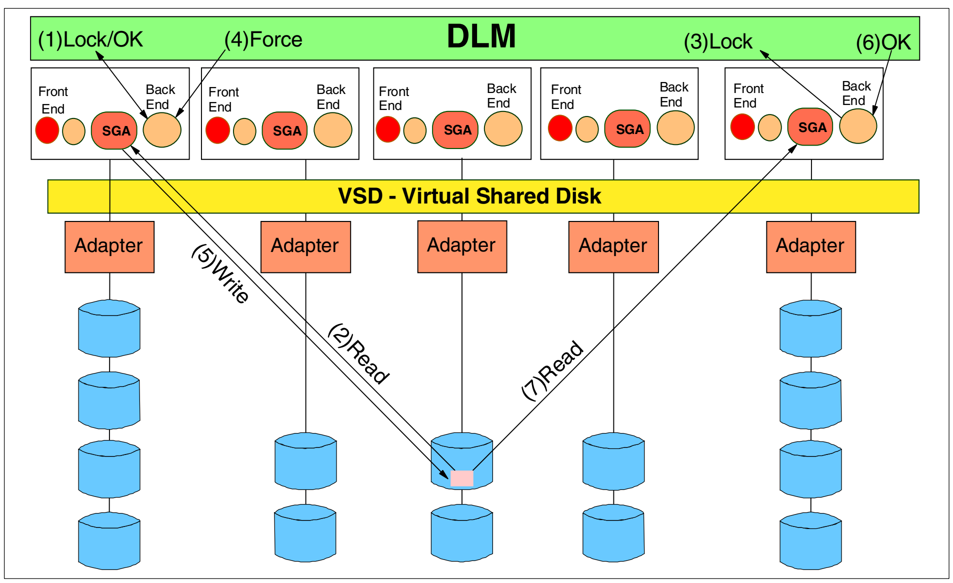

Figure 54. Distributed lock manager operation

The following is a brief description on how the DLM works by taking a simple example where the instances on the far right and far left want to update the same block in the database. The order is as follows:

1. The right instance wants the block, so it requests a lock from the DLM for that block, and the DLM grants the lock because no other instance has the lock.
2. The right instance reads the block into its local SGA buffer cache.
3. The left instance wants the block, so it requests a lock from the DLM for that block, but the DLM finds that the block is locked.
4. The DLM requests the right instance to release the lock.
5. When the right instance has finished the transaction, it writes the block to the disk via the VSD and informs the DLM that it no longer needs the lock and releases it.
6. The DLM grants the lock to the left instance.
7. The left instance reads in the block.

Note that this example is for instances needing to update a block. In the case of read access, multiple instances can have a read-only copy of a block, but if an instance wants to update the block later, they all have to release the read copy.

Improvements have been made in Oracle8*i* (Version 8.1.5 and later), which improve OPS performance by eliminating the need for two disk operations when one instance needs a read-only copy of a data block that is modified in another instances cache.

In this case, a ping is replaced with a cache to cache transfer of the data block over the switch or network. This is referred to as *Cache Fusion*. The cache to cache transfer over the network, along with the required DLM communication, is much faster than having one instance write the block out to disk and the other instance read it from disk.

17.2 Oracle tuning

In this chapter, we cover the important items to make your Oracle database run well on the AIX platform. These tuning details are AIX specific, and we do not recommend using them on other platforms.

You can work with Oracle in SP systems in two ways:

- With Oracle Parallel Server, with a parallel database, using VSD and taking advantages of the SP parallel structure. In this case, the database is based in raw devices.

- With the traditional version of Oracle in any node (or SP-attached server node). In this case, the tuning for the database and the system is the same as any other RS/6000 system. Both JFS or raw devices can be used. In this chapter, we cover the two possibilities.

17.2.1 Oracle tuning order

There are several levels that need to be addressed in order to tune the Oracle RDBMS:

1. Gather all information possible to build a picture of the machine and how it is behaving.
2. Document the Oracle parameters. This can be done in different ways using DBA tools.
3. Check for the obvious mistakes that degrade performance, in both AIX and Oracle:
   - Incorrect use of asynchronous I/O. The feature is a normal operation on AIX because it reduces CPU use with no risk and is recommended by IBM and Oracle.
   - Incorrect disk subsystem installation. Not balancing disks between SSA loops and adapters, mirror copies placed in the same adapter, etc.
   - Redo log disks should be separated out onto a dedicated and mirrored pair of disks.
   - Oracle running out of memory. It must have a significant amount of memory to operate at high performance level.
4. Tune the database and system for maximum performance, reviewing these areas:
   - Disk and I/O: Sections from Section 17.2.2.1, “AIX asynchronous I/O” on page 442 to Section 17.2.2.6, “Disk sets for hot disk avoidance” on page 445 cover these items.
   - Memory: Sections from Section 17.2.2.7, “AIX sequential read ahead” on page 446 to Section 17.2.2.10, “AIX buffer cache size” on page 447 cover this item.
   - CPU: Go to Section 17.2.2.11, “SMP balanced CPU utilization” on page 449 for more information.
   - Network: Go to Section 17.2.2.12, “Networking parameters” on page 450 for more information.
   - Access method (SQL)
   - Oracle contention (processes fighting for resources)

These last two items are only Oracle-related and will not be cover in this redbook.

17.2.2 Tuning AIX for Oracle

This section contains the most common AIX tuning hints that should be considered as normal operation. They are not in any particular order.

17.2.2.1 AIX asynchronous I/O

In the Oracle init.ora configuration file, set this to TRUE. For Oracle 7:

```
use_async_io = true
```

and for Oracle 8:

```
disk_asynch_io = true
```

Then set the minservers and maxservers using the AIX `smit` command SMIT->Devices->Asynchronous I/O->Change/Show Characteristics of Asynchronous I/O (or just type `smit aio`) to:

- MaxServers = 10 * number of disks, but with a maximum of 10 times the number of processors in the machine.
- MinServers = MaxServers /2

For example, on a machine with 30 disks, Max Servers = 300 and MinServers should be 150, but if this machine only has four CPUs, then MaxServers = 40 and MinServers = 20 is sufficient. Higher numbers will not hurt performance, as it only results in more kernel processes running that do not actually get used. See Section 6.3.11, “Asynchronous I/O” on page 169 for more details.

Using asynchronous I/O is likely to increase performance. There is no risk of reducing performance, so it should always be used.

17.2.2.2 AIX Logical Volume Manager versus Oracle files

To spread out the data and disk workload across disks, you have two choices:

1. Use Oracle - As the Oracle DBA, use a lot of Oracle files and place them onto disk by creating a logical volume on a particular disk, then monitor the database objects (tables and indexes) in the files and move objects between files to balance disk use. In this case, the AIX System Administrator simply creates each Oracle file (JFS or raw logical volume) on a separate disk.
2. Use AIX - As the AIX system administrator, use the AIX Logical Volume Manager (LVM) to create each JFS or raw logical volume on multiple disks. Use striping to spread data across disks and then monitor the AIX

physical volumes. In this case, the Oracle DBA does not have to balance disk use between Oracle files and, therefore, disks.

Note

It is strongly recommended by IBM and Oracle that the benefits of the AIX LVM are fully used.

The AIX LVM has a lot of options you can tune, and striping data across disks is very effective, as it makes full use of the disks in terms of usage, makes excellent use of read-ahead for sequential I/O, and spreads disk I/O evenly for better performance. For striping, you can use the following values to start:

- Stripe unit size = 32 KB or 64 KB
- max_coalesce = 64 KB
- minpgahead = 2
- maxpgahead = 16

Some recommendations about stripping in AIX:

- The striped LV size must be a multiple of the number of drives used. For example, if the strip size is 32 KB, and the LV is spread across eight disks, then the size of the LV must be a multiple of 32 * 8 K. This allows the LVM to create the LV with a complete number of stripes and avoid a situation where the last stripe does not cover all the disks.
- It is recommended that striped data and the database logs are on different sets of disks.
- After AIX 4.3.3, the LVM allows striping and mirroring at the same time. This has been tested for performance and works well.

Benchmarks have shown using AIX LVM to stripe data can increase performance on disk bound systems up to three times normal. Disk bound batch workloads particularly benefit from this.

17.2.2.3 Create logical volumes at a standardized size

When creating logical volumes in order to use them as Oracle files, create all of them with standard size.

We recommend making all the files 1 GB or 2 GB on large machines. This size of file does not cause large file problems and can be copied or moved around the system. It is worth staying under the 2 GB limit because larger files can still cause problems and lower performance. Some applications, tools, and commands have problems with files larger than 2 GB because they have not been re-coded to make use of the 64 bit file system calls required to handle larger file sizes. To hold files larger than 2 GB in a JFS requires the use of the *large file support* version of the JFS. This is not as efficient as the regular JFS. There are occasions when smaller files are needed in the database, and we recommend using 64 MB as a minimum.

17.2.2.4 AIX JFS versus raw devices

This is a much discussed subject with arguments in favor of both sides. The two options are:

- JFS - If your database is not I/O bound, that is, your applications do a lot of computation on small or infrequently retrieved data, then JFS is a good choice, because its simpler to create, work with, administer, and back up/recover.

  Reading a block from a JFS file means it is moved from disk to the AIX buffer cache or file systems cache and is then copied from the file systems cache to the Oracle SGA for the process to access the data.

- Raw devices are harder to work with, as they do not appear in the file system. Raw devices avoid the double buffering of data. This means the data is read from disk straight to the Oracle SGA. The same is true for writing to a raw device. Because the database does not use the AIX buffer cache for data access, this cache size can be a lot smaller, and memory is freed up and can be used to support a much larger Oracle SGA.

In Table 46, you can see the common memory distribution for JFS or raw device databases:

Table 46. Memory distribution for JFS and raw device databases

JFS database	raw device database
33% for processes	33% for processes
33% for SGA	60% for SGA
33% for file systems cache	6% for file systems cache

Raw devices are faster. The number of databases that cannot benefit from faster disk access is small. Therefore, raw devices are recommended for performance reasons. The main problem is that some backup systems do not directly support the backing up of raw devices.

From benchmark experience, moving to raw disks for disk I/O bound systems is likely to increase performance by 0 - 50 percent provided the SGA size is also adjusted to counteract that the AIX buffer cache is no longer used.

Note

The move from JFS to Raw devices is a simple process and does not require exporting and importing the whole database, but can be done file by file.

17.2.2.5 AIX disk geometry considerations

With the AIX LVM, you can place data (logical volumes for JFS use or raw devices) on particular positions of the disk. The center part of the disk is the fastest because, on average, the seek time to the center position is less than a seek to or from the edges of the disk, which causes higher head movements. Use the -a option of the `mklv` command to set this or, in SMIT, the `POSITION on physical volume` option.

If the disk center position is the only part of the disk that is used, then the disk average seek times will be much faster. This means the disk will appear to have lower seek times than the average quoted for the entire disk. This option should be considered if you have highly performance critical parts of the RDBMS where maximum performance outweighs the extra costs.

This may increase performance by up to 10 percent.

Logical Volume placement

When creating a database, create the performance critical logical volumes first in the center of the disks to ensure they get the best possible performance.

17.2.2.6 Disk sets for hot disk avoidance

This redbook consistently recommends disk protection and avoiding hot disks within the database.

For both RAID 5 and striped mirror disk protection, the data is spread across multiple disks:

- For RAID 5, we recommend seven disks plus one parity disk for an eight disk RAID 5.
- For striped mirror (both PP level or fine stripe on AIX 4.3.3), we recommend spreading across eight or 16 disks.

The result is that a single hot disk is impossible, but you can still have hot eight or 16 disk sets. But, the performance problems will be eight (or 16) times smaller as a result.

It is traditional, and Oracle's recommendation, to split index, data, and temporary tablespaces onto different sets of disks. This means we should have these groups of eight (or 16) disks assigned to one purpose or another.

For more information about I/O tuning go to Section 6.3, "I/O" on page 141.

17.2.2.7 AIX sequential read ahead

This only affects JFS file system based database files.

The Virtual Memory Manager spots sequential reading of JFS based database files by watching the access pattern of read system calls. After a number of sequential reads are noticed, it will attempt to read up to the maxpgahead blocks of the file in advance. By default these, are:

- minpgahead 2
- maxpgahead 8

These can be increased to increase sequential reading ahead of data, which will speed up sequential reads using `vmtune`. For example:

```
vmtune -r 512 -R 1024
```

Keep the numbers in powers of 2.

For more information about memory tuning, see Section 6.1, "Memory" on page 115.

17.2.2.8 AIX paging space

Paging in any UNIX is very bad news because it can seriously reduce performance. If paging gets bad, AIX will start swapping processes out. Ideally, paging should be completely avoided. When users start and stop large processes, there is likely to be some limited paging (this is how AIX gets the program into memory), but this paging should be limited and short lived, and no paging for long periods is the best. We recommend, in general, spreading out paging space onto multiple disks.

Check to see what is in memory with `ipcs` and `ps aux` (check the RSS column values) and use the `vmstat` command to monitor paging. See Section 6.1, "Memory" on page 115 for more details, or the AIX manuals.

In large configurations, such as the RS/6000 S Series (SP-attached server node) with 12 to 24 CPUs, AIX can support thousands of users and their processes. With large numbers of programs running, there is a high chance that they are going to change their working sets. This causes regular paging. So, for this size of machine, benchmark people use the rule 10 pages per second per CPU. So, for a 12-way machine, 120 pages per second is acceptable; zero is preferred.

17.2.2.9 AIX free memory

For JFS database files, there can be a copy of the disk block in both the Oracle SGA buffer cache and in the AIX buffer cache. This double buffering can affect performance and cause disk I/O bottlenecks. There are two AIX buffer cache tuning parameters that determine the size of the free list:

- minfree - Below this limit, page stealing starts trying to reclaim memory pages.
- maxfree - Above this limit, page stealing stops.

For a full explanation of these parameters, see the AIX documentation or Section 6.1.2.4, "minfree and maxfree" on page 119.

On machines with large memory (1 GB or more), you should try to keep a small amount of free memory. Making minfree and maxfree larger should increase the free memory slightly. This means always wasting a little memory, but also means disk I/O is not delayed. For example, keep 128 pages free by using:

```
# vmtune -f 128 -F 144
```

On machines with less memory (less than 1 GB), do not change these parameters.

We recommend that only experienced AIX administrators change these limits because if they are set wrong, they can cause a system to perform slowly or strangely.

17.2.2.10 AIX buffer cache size

This depends a lot on the workload and I/O characteristics of your database and whether you are using a JFS file system based database or raw devices.

There are two AIX buffer cache tuning parameters that determine the AIX buffer cache size:

- minperm - Below this limit, file and code pages are stolen.
- maxperm - Above this limit, only file system pages are stolen.

The numbers are pages of memory used by the buffer cache; the system will try to keep the AIX buffer cache size between minperm and maxperm percentage of memory. Use the `vmtune` command with no parameters to determine the current values of the minperm and maxperm. At the bottom of the output is the total pages in the system. Look for:

```
number of valid memory pages = [number]
```

Also, at the bottom of the output is the percentages of memory that the values of minperm and maxperm work out too, which is often more helpful. To change minperm and maxperm, you have to work out the actual number of pages that will work out to the percentages you are aiming for.

For a full explanation of these parameters, see the AIX documentation or Section 6.1.2.6, “minperm and maxperm” on page 123.

The defaults should work out to approximately 20 percent and 80 percent of memory respectively.

Buffer cache size for a JFS based database

For JFS based databases, you are using the AIX buffer cache.

On machines with large memory (1 GB or more), you will find that 20 percent of memory is not available for file system cache (200 MB). This is a large amount of memory. There are two cases to consider:

- On systems that have only a few or small applications, this memory is not all used up by the application or RDBMS code. In this case, raising maxperm will make more of this memory available for AIX buffer cache use (for example, change maxperm to 95 percent of memory).
- On systems with very large applications, you might find that AIX keeps stealing application code pages, which results in continuous paging of application code. In this case, lowering maxperm will allow higher percentage of memory to be allocated to application code and reduce paging (for example, change maxperm to 70 percent of memory).

On machines with less memory, do not change these parameters or be very careful, as the default values have been found to work well.

Buffer cache size for a raw device based database

For raw device (also called raw disk, partition, or logical volume) based databases, you are not using the AIX buffer cache to any great extent. It is actually being used for disk I/O for AIX processes and, for example, RDBMS error log files, but the bulk of the RDBMS disk I/O is bypassing the AIX buffer cache (that is a major point of using raw devices).

On machines with large memory (say 1 GB or more) that are only running an RDBMS, you will find that between 20 percent and 80 percent of memory has been earmarked for the AIX buffer cache. But the Oracle SGA is occupying a large part of memory. For example, the SGA might be 50 percent of memory; so, the other 50 percent is used shared between processes and buffer cache. This means the values of 20 percent to 80 percent are not sensible settings. We might page out process memory (code or data) from memory unnecessarily.

When the system is running normally, use the `vmtune` command with no parameters to determine the amount of memory used for the buffer cache. At the end of the output, you will find a line like:

```
number of file memory pages=[number] numperm=[num] percent of real
memory
```

The second number (the numperm percentage) is the one to think about. If this is less than the minperm (on the line above), then we have the memory pages being stolen from code and file system buffer cache equally. This probably results in unnecessarily paging out processes.

Another way to look at this is to say that the file system buffer cache should be 20 percent to 80 percent of the memory *not* occupied by the Oracle SGA. If, for example, the SGA is 50 percent of memory, then the buffer cache should be 10 percent to 40 percent. The important value is that of minperm. You could never reach the 80 percent default value of maxperm because the SGA is taking up a large part of memory.

There are a number of cases to consider:

- If minperm is greater than 20 percent of the non-SGA memory, set minperm to be this value and reconsider once the system has settled down.
- If the numperm number is greater than minperm, you should consider allocating more memory to the Oracle SGA.
- If numperm is smaller than minperm, you are freeing process memory and should reduce minperm. For example, change minperm to 2 percent of memory.

17.2.2.11 SMP balanced CPU utilization

Most large systems are now SMP machines, as this allows multiple fast processors to scale up to higher power machines. There is one small drawback in that the application and database must have enough processes to keep all the CPUs busy. If only one process is used, only one CPU can be used (assuming it is a single threaded application), and, therefore, only a fraction of the available power of the machine is used. Fortunately, Oracle has multiple processes (one per user as a default) and for large tasks allows it to be parallelized to make it effective on an SMP. It is worth checking that this is actually happening, particularly for batch jobs that have, in the past, often been implemented as a single process. Use the `sar -P ALL 1 10` command to check if all CPUs are busy. For more information about CPU tuning, go to Section 6.2, “CPU” on page 129.

17.2.2.12 Networking parameters

For a complete description of this topic go to Chapter 4, “Network tuning” on page 21 or more specifically to the section that describe the network configuration for database environments, Section 4.7.3, “Tuning for commercial and database environments” on page 89.

17.2.3 Top 10 Oracle parameters

The parameters that make the biggest difference in performance (and should, therefore, be investigated in this order) are listed in the following sections.

17.2.3.1 db_block_size

This cannot be changed after the database has been created. It is vital to get this correct before you start. As AIX does all I/O at a minimum of 4 KB, we do not recommend any sizes that are smaller than this, as it will be slower.

We suggest the following Oracle block sizes:

`db_block_size=4096` for:

- Small databases (less than 10 GB)
- JFS based databases (because AIX does 4 KB pages)
- OLTP workloads, where you typically only want one row in the block and reading an extra block would not help
- Mixed workload (OLTP and DSS) databases to assist OLTP performance

`db_block_size=8192` for:

- Large database (greater than 10 GB)
- DSS workload, where reading more rows in one go can help
- Large rows, where most of the rows of the database are large, and you will get less wastage at the end of blocks with larger blocks
- Databases with batch workloads, where the dominant database load involves using large table scans (and not indexed single row accesses)

`db_block_size=16384` for:

- For very large databases (greater than 200 GB) with DSS workloads

17.2.3.2 db_block_buffers

This value is the number of disk blocks stored in the SGA. This is the largest part of the SGA. The memory size allocated will be:

```
db_block_buffers * db_block_size
```

If you have no idea how large to set the parameter (which is typical on a new system until you start running tests), then set it so that the SGA is roughly 40 percent to 50 percent of memory.

If your database data is in a Journaled File system (JFS), then you will need to allocate space in real memory for the AIX buffer cache. The AIX buffer cache is dynamically controlled, but you should make sure sufficient memory is available for it. However, if your data is held on raw devices, then disk I/O does not use the AIX buffer cache, and it does not need to be large, and more memory can be allocated to the Oracle buffers. Figure 55 shows this difference.

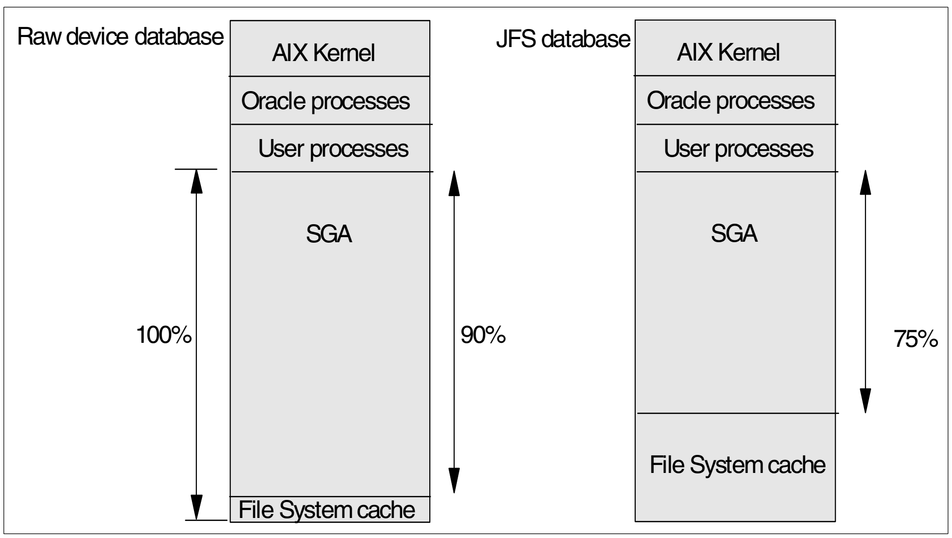

Figure 55. SGA buffer cache memory sizes for raw device and JFS DBs

If you can estimate how much memory is left after allowing for

- AIX (32MB)
- The basic Oracle process

- The Oracle Servers
- The user application processes

then we recommend to allocate the following amount of the remaining memory:

- For JFS based databases, 75 percent
- For raw devices, 90 percent

Once the system is running, make sure it is not paging (in this case reduce the SGA size) or has free memory (make the SGA larger). The aim with the number of buffers is to have a very high buffer cache hit ratio (greater than 95 percent).

17.2.3.3 use_async_io or disk_asynch_io

AIX fully supports the asynchronous I/O for both JFS and raw devices.

On Oracle 7, this parameter is called *use_async_io*.

On Oracle 8, this parameter was renamed to *disk_asynch_io*, and the default is set to TRUE.

17.2.3.4 db_writers, db_writer_processes and dbwr_io_slaves

In Oracle 7, the name of the parameter is *db_writers*.

This parameter decides how many database writer processes are used to update the database disks when disk block buffers in the SGA are modified. Multiple database writers are often used to get around the lack of asynchronous I/O in some operating systems, although it still works with operating systems that fully support asynchronous I/O, such as AIX.

We have recommended, in the previous section, that asynchronous I/O is used, so we recommend you use a single database writer to keep things simple. Therefore, set this parameter to 1 to reduce the database writer overhead.

In Oracle 8, this functionality is covered by two parameters called *db_writer_processes* and *dbwr_io_slaves*.

The db_writer_processes parameter specifies the initial number of database writer processes for an instance. In most cases, one database writer process is enough, but it is recommended to use multiple processes to improve performance for a system that writes a lot of data. The dbwr_io_slaves parameter is similar to the Oracle 7 db_writers parameters. However, if the dbwr_io_slaves parameter is used, then only one database writer will be used regardless of the setting for db_writer_processes.

Therefore, set the db_writer_processes parameter to 1 to reduce the database writer overhead and leaving dbwr_io_slaves at the default value.

17.2.3.5 shared_pool_size

This parameter is very hard to determine before statistics are gathered about the actual use of the shared pool. Its size can vary from a few MBs to very large, like 100 MB, depending on the applications' use of SQL statements.

If you have no statistical information, use the following defaults to create a base of information:

- For smaller systems (128 MB to 512 MB of memory)

  ```
  shared_pool_size = 3 MB
  ```

- For system with more than 1 GB

  ```
  shared_pool_size = 30 MB
  ```

Some applications that make heavy use of the this area have a shared_pool_size of up to 200 MB.

17.2.3.6 sort_area_size

This parameter sets the size of memory used to do in-memory sorts. In OLTP environments, sorting is not common or does not involve large numbers of rows. In Batch and DSS workloads, this is a major task on the system, and larger in-memory sort areas are needed. Unlike the other parameters, this space is allocated in the user process and not just once in the SGA. This means that if 100 processes start a sort, then there is 100 times sort_area_size space allocated in memory, so you need to be careful, or you will run out of memory and start paging.

Set sort_area_size to be 200 KB on a small system with a lot of users, and at 2 MB on larger systems.

17.2.3.7 sql_trace

This parameter makes Oracle collect information about performance. This creates an extra load on the system. Unless you require the information for tuning, this should be set to FALSE.

17.2.3.8 timed_statistics

This parameter makes Oracle collect timing information about response times, which creates an extra load on the system. Unless you require the information for tuning, this should be set to FALSE.

17.2.3.9 optimizer_mode

If the application provider recommends to set this to RULE, do so. Otherwise, you should set this to CHOOSE (which is the default for newer versions of Oracle).

17.2.3.10 log_buffer

The redo log buffer is used to store the information to be sent to the redo log disk. This buffer speeds up the database performance by allowing transactions to record the updates to the database but not send nearly empty log records to the redo log disk. If many transactions added to the log buffer faster than they can be written to disk, then the buffer can get filled up. This is very bad for performance.

We recommend a minimum of 128 KB, but many DBAs just set this to 1 MB to make it extremely unlikely to ever fill up.

17.2.3.11 rollback_segments

The rollback segments contain the original contents of blocks that are updated during a transaction.

We recommend a lot of small roll backs. For systems with less than 32 active concurrent transactions at one time that are updating rows, create eight rollback segments. However, working out the number of transactions in advance is nearly impossible. If you have higher numbers of active transactions, then Oracle recommends one rollback segment per four transactions, but do not go higher than 50 rollback segments.

Chapter 18. Lotus Notes

Lotus Domino Server and Lotus Notes Workstation is a client/server environment that allows users (or clients) to communicate securely over a local area network or telecommunications link, and create and access documents residing on a shared computer (or server). With Lotus Domino Server and Lotus Notes Workstation, people can work together regardless of their software or hardware platform or technical, organizational, or geographical boundaries. Lotus Notes Workstation combines an application development environment, a document database, and a sophisticated messaging system, giving you the power to create custom applications for improving the quality of everyday business processes in areas such as product development, customer service, sales, and account management. At its most basic level, Lotus Notes Workstation is a document database, serving as a repository for both textual and other information, for example, images, presentations, and spreadsheets. Lotus Domino Server and Lotus Notes Workstation provide the ability to distribute this information throughout an enterprise via replication, yet only those who need to see the information have access to it. In short, the intent is to improve Communication, Coordination, and Collaboration across any enterprise.

The Domino Server It provides services to Notes Workstation users and other Domino Servers, including storage and replication of shared databases and mail routing. The Lotus Domino Server can run on PCs under OS/2 and Windows NT. It can also run as a NetWare NLM, or under UNIX systems such as IBM AIX, HP-UX and Sun Solaris.

Only the TCP/IP (Transmission Control Protocol/Internet Protocol) and IPX/SPX (Internetwork Packet eXchange/Sequenced Packet eXchange) network protocols are supported for Lotus Domino Server Release 4.5 running on AIX.

18.1 What can be tuned now in the SP and AIX environments?

The first recommendations to tune an AIX based machine, like our SPs, is to upgrade the hardware.

- Add more memory (256 MB for 300 Users).
- Add L2 cache (if possible).
- Install a faster processor.
- Add additional processors on SMP systems.

- Use fast disk - Raid 0 with striping

That looks very easy from the software point of view and totally opposite to what we mentioned in Chapter 15, “Web applications” on page 393. There we talked about the yearly hardware performance increase compared with the workload increase. But Lotus Notes is usually used in companies where the number of employees is not growing 1000 percent a year. The price for hardware is also getting cheaper, and in this particular case it is easier to increase system performance by adding/buying new hardware. But we do have some possibilities on the AIX side to tune some performance related values:

- Use IPX/SPX instead of TCP/IP if you don't need TCP/IP functionality such as IP-routing and so on.
- Use `smit` to increase the maximum number of processes allowed per user to 102.
- Use `smit` to increase paging Space in the range of 300 to 500 MB.
- Use `smit` to set the maximum number of Licensed Users to 16.
- Set the *SHARED_DPOOL* variable as followed:

  `export SHARED_DPOOL=`*(1/4(Total Machine RAM in bytes)/10)*

18.2 Transactional logging and how it operates

Transactional Logging is a new feature in Domino R5. It is essentially a new method of writing out database changes to improve performance and to ensure data integrity. Its main purpose is three-fold:

- To improve performance on the R5 server through sequential writes to the Transactional Logs.
- Better data integrity by avoiding inconsistencies and data corruption
- Faster server restart and crash recovery

A Transactional Log is simply a binary file where transactions are written. The transactions are saved in log extents that have a TXN extension. Each log extent can grow to a maximum capacity of 64 MB before a new extent is created and then written to. Multiple log extents collectively can grow to a maximum size of 4 GB. The number of log extents created depends on the setting specified in the Server document in the Maximum Log Size field.

18.2.1 Performance improvement

When transactional logging is enabled on the server, the performance improvement is mainly due to the nature of how transactional logging operates. The writes to the Transactional Log are sequential. This is faster since there is less head movement and there is never a need to search for a place on the disk to write as there is in R4 or if transactional logging is not enabled.

The Transactional Logs must be on a separate physical drive for there to be any performance improvement. It is not sufficient to simply redirect the logs to a separate partition or a separate logical drive. In general, if the transactional logs are on a separate drive, a 10-20% improvement should be seen. However, if the logs are put on the same drive, it is likely that there will be approximately a 60% degradation.

In R4, writing to disk was time consuming. Modifications could occur across multiple databases or different parts of one database. As a result, the head had to move over various areas of disk to change or update data. This means there was a significant amount of transaction time committing data to the actual NSF (database). Without the benefit of transactional logging in R4, fixup relies on the fact that 99.9% of the data is present in the NSF to correct integrity problems.

In R5, when transactional logging is enabled, complete transactions are "committed" to the Transactional Log. All writes are done to the transactional log before they are ever written to the database. The writes are done sequentially at least after each transaction so the Transactional Log is up to date generally to the hundredth of a second. Again, because the writes are sequential, there is less I/O and performance is improved. Please note that view indexes and attachments are never transactionally logged.

When transactional logging writes transactions to the logs, an undo record and a redo record are usually committed for each transaction. First an UNDO log record is generated in the event of a system outage. This is done before a change is written to a database. If a transaction is undone, a change is never made to the actual NSF. Before committing a transaction, a REDO record is also generated. It is used to re-apply a transaction from the transactional log to the database in the event that it did not get flushed to the NSF before a server outage. Undo and redo records ensure that if a change is half done it will be fully undone, and if a change was completely done then it will be fully re-done to the NSF.

18.2.2 Flushing and hardening

Once changes are put into the Transactional Log, the changes must also eventually be hardened to the database. This occurs through a process called flushing. Any open database has an in-memory version of the database that is held in the UBM (Unified Buffer Manager). Flushing moves all changes that were made to the database but only kept in memory (UBM) to the actual NSF file. There is no set interval for this as the UBM determines when flushing will occur. It is usually done when there is a lull in the server activity. The DBIID (Database Instance Identifier) is used to correlate the updates in the Transactional Logs and in-memory to the respective database. It is important to note, however, that the Transactional Logs are not read from during this process because the Transactional Logs are mainly a write-only object. The updates are read and flushed from the UBM. They are only read from the Transactional Logs during crash recovery. Transactional logging is more expedient because there are not a lot of read/writes to it during server production. Otherwise performance would suffer and it would defeat one of the purposes of transactional logging.

The Runtime/Restart Performance field in the Server document determines how many MB of changes are kept in memory. The amount of space used is bound by the "performance" level chosen in the Server document. There are three choices: Standard (default), Favor Runtime, and Favor Restart Recovery Time. If Standard is selected, the Redo. Limit is 49 MB. This means that checkpoints during runtime are minimal and 49 MB worth of changes are held in the UBM before they are flushed and hardened to databases. The Favor Runtime choice has a Redo. Limit of 500 MB. This means that more information is held in the UBM and hardened to the database less frequently. There are also fewer checkpoints, making runtime faster but server startup slower. The Favor Restart Recovery Time choice allows for more checkpoints during runtime. There is less information held in the UBM and data is hardened to databases more frequently. The trade-off is that production time is slower but server restart is faster.

18.2.3 Crash recovery

After a server outage the Transactional Logs are played back. The Recovery Point is used to determine the oldest log information that needs to be re-applied to databases. It is the point that databases are restored to, which is usually only a few milliseconds before the outage. Partial transactions will be undone and rolled back to the last good state in an effort to avoid corruption in the database. They will not be hardened to the database. It is important to realize that transactional log recovery can only re-apply or undo transactions not written to disk at the time of the failure. Anything that has already been hardened to a database is not touched by transactional logging.

Customers should not move databases away from a server and copy them over from another server after a crash. If this is done, the DBIID will change and information in the Transactional Log will not be applied to the database (loss of data) because the DBIID in the Transactional Log will be inconsistent with the DBIID in the newly copied database. If the DBIID changes and a backup is not taken after the fact, the database cannot be successfully restored (the backup will have the old DBIID and the transactional logger will not "know" the old DBIID).

18.2.4 Transactional Logging NOTES.INI Parameters

Transactional logging is enabled in the Server document. All the fields in the Server document map to specific NOTES.INI parameters.

The parameters are as follows:

- **TRANSLOG_AutoFixup=0**
  Tells whether autofixup is Enabled or Disabled.
- **TRANSLOG_UseAll=0**
  To use all available space or not.
- **TRANSLOG_Style=0**
  Circular vs. Archive.
- **TRANSLOG_Performance=2**
  Favor runtime, Standard, Favor restart recovery.
- **TRANSLOG_Status=1**
  Whether transactional logging is Enabled or Disabled.
- **TRANSLOG_Path=XXX**
  Specifies the path to the .TXN files.

18.3 Tuning servers for maximum performance

This section highlights Lotus Notes server tasks and discusses common server performance problems.

18.3.1 Server tasks

Many server tasks are started by default. If you don’t need a particular task it is much better to turn it off; tasks such as Calendaring and Scheduling can require up to 15 to 20 percent of the CPU. The table below summarizes the main tasks you are able to disable:

Table 47. Server tasks that may be disabled

Task Name	Turn it off if
Router	You are not using the server for electronic mail or workflow.
Calconn Sched	You are not using the server for calendaring and scheduling.
AMgr	You do not run schedules agents. Agent Manager is not required for WebQueryAgents.
Collector Reporter	You do not want to track server statistics at regular intervals. You can still generate server statistics on demand.

18.3.2 Server performance problems

The following are the most frequent performance problems seen when running a Lotus Notes server. Most are solved by reconfiguring parameters in the server notes.ini file.

18.3.2.1 It takes a long time to display views

Large view will take longer to display, but being able to cache these views in a larger view buffer will speed performance as the server will not have to read the view from the disk.To increase the view buffer you need to increase the NSF_BUFFER_POOL_SIZE parameter in the notes.ini file. This setting increases the maximum size of the cache for storing views (in bytes). You should also see a corresponding decrease in I/O activity. We do not recommend increasing it beyond 1/4 the physical memory.

18.3.2.2 It takes a long time to access databases

If database access is slow you can increase the `NSF_DBCACHE_MAXENTRIES` parameter. This setting determines the maximum number of database open/close handles in the cache. You can increase this up to 1872 entries. This parameter is also found in the notes.ini file.

Large databases require more I/O and use more memory so upgrading the I/O subsystem or adding more memory will improve the performance. If this is not possible another solution would be to archive old or little used parts of the database to reduce its size.

To increase database performance you can also split users across multiple replicas to save burdening one database and server with all users. If you are using a single server you can put the highly used database on a different hardware array and refer to it via a database link.

Using clusters can help too. They provide a means for load balancing heavily used servers and/or databases. Key parameters here include Server_Availability_Threshold and Server_MaxUsers.

18.3.2.3 CPU saturated

If you are seeing CPU bottlenecks being caused by Notes processes, it could be an indexer problem. First, increase the number of indexer tasks (this is only relevant if you have a multiprocessor system); multiple processors can handle more than one indexer task and so the views are more likely to be up to date. A good guideline is to set the number of indexers to the number of CPUs minus one. For example if your server has 4 processors, use 3 indexers. Be careful not to run too many indexers, as this will prevent other server processes from getting to the CPUs.

You can increase the notes.ini file setting UPDATE_SUPPRESSION_TIME. This determines the minimum amount of time that must pass before the indexer runs again.The default is 1 (the unit for this setting being in minutes). Increasing this value means that the indexer will run less frequently, freeing up server resources.

The UPDATE_SUPPRESSION_LIMIT parameter determines at what number of view update requests the view will be updated and the indexer run. The LIMIT parameter can override the TIME parameter if the LIMIT number of parameters are received.

Chapter 19. Communication server

This document contains tips for network tuning in an SNA environment. It applies to AIX Version 4.3.

For more information on network tuning in an SNA environment, refer to publications *Communications Server for AIX V4R2 Planning and Performance Guide*, SC31-8220 and *IBM eNetwork Communications Server for AIX: Understanding and Migrating to Version 5: Part 2 - Performance,* SG24-2136.

19.1 Communications buffers (mbufs)

Communications Server uses mbufs to send and receive data across the network, but it is just one of the subsystems that uses mbufs. Communications Server mbuf resource utilization can affect the performance of other subsystems such as TCP/IP, NFS, and AFS.

There are two ways to determine how much memory should be made available as communications memory buffers (mbufs). This amount of memory is defined using the maximum number of buffers or `maxmbuf`.

1. 4096 * # of communications adapters

   This method counts the total number of communications adapters in the RS/6000 (whether being used for SNA or not), and then multiplies this number by 4096. The result is used as the setting for `maxmbuf`.

   For example, if there are two Token-Ring adapters that are being used for SNA and two Ethernet adapters that are being used for TCP/IP, the calculation would be four adapters multiplied by 4096, or 16384. 16384 would be the number used for `maxmbuf`.

2. 25 percent of real memory

   SNA generally requires more memory for buffers than other applications. With the larger memory sizes that have become available, SNA will usually operate efficiently if the `maxmbuf` parameter is set to a number equal to 25% of real memory. In most circumstances, any number larger than 64 MB (65536) will not yield any appreciable benefit.

   For example, if there is 128 MB of real memory on the machine, set the `maxmbuf` parameter to 32 MB (32768). If there is 512 MB of memory, set `maxmbuf` to the recommended maximum of 64 MB (65536).

   If SNA_SRF errors are being logged in the system error log, this parameter should be checked.

19.1.1 Setting maxmbuf

To check the current setting of this parameter, enter the following:

```
# lsattr -El sys0 -a maxmbuf
```

To change this parameter, enter the following:

```
# chdev -l sys0 -a maxmbuf=new_value
```

Note

AIX 4.3, the default setting for the sys0 attribute `maxmbuf` is 0. This value is known to cause problems with SNA. Set `maxmbuf` to any value other than 0.

This should be the same as the current setting for the no command's thewall setting.

19.2 SNA DLC link parameters

Certain SNA DLC profile parameters significantly impact throughput of file transfer applications. For Token-Ring and Ethernet, use the following guidelines:

- The *Transmit_Window_Count* parameter on any node A should always be greater than the *Receive_Window_Count* on any node B for any two nodes that are connected to each other. This is true even if a node is not an RS/6000.
- Using large values for Transmit_Window_Count and Receive_Window_Count does not enhance throughput. Setting the Transmit_Window_Count to 16 and Receive_Window_Count to a lesser value should be sufficient for peak throughput.

For an SDLC connection, the Receive_Window_Count does not exist. With SDLC, the primary_repoll_time_out parm setting must allow enough time for the largest frame to travel to and from the secondary station. Otherwise, unnecessary polls will flood the link, eventually bringing it down.

If the link speed is X bits per second and the largest frame is Y bits, then the primary_repoll_time_out should be larger than 2(Y/X).

19.2.1 LAN-based DLC characteristics

The Token-Ring, Ethernet, and FDDI Data Link Controls contain a parameter called the *Depth* of receive queue. This queue is logically between the device handler and the data link control, and holds asynchronous event notifications such as the receipt of data packets. This queue defaults to a depth of 32 entries and can be changed by the operator if log entries indicate that the device handler is overrunning the DLC due to heavy system activity. This attribute should be changed in small increments because it directly affects system storage availability.

To change this value, use the following *fastpath* command:

```
# smit devices
```

Then follow these steps:

1. Select Communication.
2. Select the type of communication from the list displayed.
3. Select Services.
4. Select Data Link Control.
5. Select Change/Show Data Link Control.
6. Change the parameter value.

19.3 Memory

A rough estimate of the amount of memory required to run a certain number of sessions is that one additional megabyte of memory will support about seven additional sessions. The exact number will depend on traffic.

19.4 Paging space

It is advisable to keep the paging space used below 75 percent to ensure that SNA resources will be available. If these resources must be "paged" out of real memory, it is important that enough space is available to prevent SNA from experiencing errors that could result in total failure of SNA communications. If the percent used is over 75 percent, additional paging space should be added.

To display total paging space and the approximate amount used, enter:

```
# lsps -s
```

To display the amount of memory, enter:

```
# bootinfo -r
```

19.5 File transfer applications

When using file transfer applications, the SNA LU 6.2 Mode Profile can be configured to maximize throughput. Use the following guidelines.

- Larger request/response unit (RU) sizes generally correspond to higher throughput. The following formula for RU sizes will provide optimum throughput in most cases.

  RU = (I-Field size) - (9 bytes)

- It is also very important to try to match the RU sizes on both sides of communicating nodes, regardless of the platform. An RU size larger than one calculated from the formula will result in segmentation of the data and cause a slowdown in data transfer.

  To find the I-field size, use the following command while the link station is active:

  ```
  # sna -d l -o long |grep -p linkstation_name
  ```

  Look for the field labeled "Max frame data (BTU) size". This I-field size is controlled from within the SNA DLC profile in the SNA configuration. The I-field size should be system defined or matched exactly with the other node configuration.

- Generally, increased throughput is also related to larger SEND and RECEIVE session pacing window sizes. Try setting the SEND pacing and RECEIVE pacing window sizes to the maximum value of 63.

  When writing file transfer applications, use large buffers in the TP code. A 16K buffer size will speed up the transfer.

Part 5. Appendices

Appendix A. Performance toolbox for AIX and parallel extensions

The Performance toolbox for AIX (PTX/6000) is a tool to monitor and tune system performance, and the Performance toolbox Parallel Extensions (PTPE) is an extension of the Performance Toolbox for use in an RS/6000 SP complex.

A.1 Performance toolbox for AIX (PTX/6000)

PTX/6000 uses a client/server model to monitor local and remote systems. It presents the performance data graphically.

Figure 56 on page 469 illustrates the client/server model used by PTX/6000. The server requests performance data from the nodes or other networked computers. The nodes or networked computers in turn supply a stream of performance data. The server displays the performance data graphically. When monitoring is complete, the server informs the nodes or networked computers to stop streaming performance data.

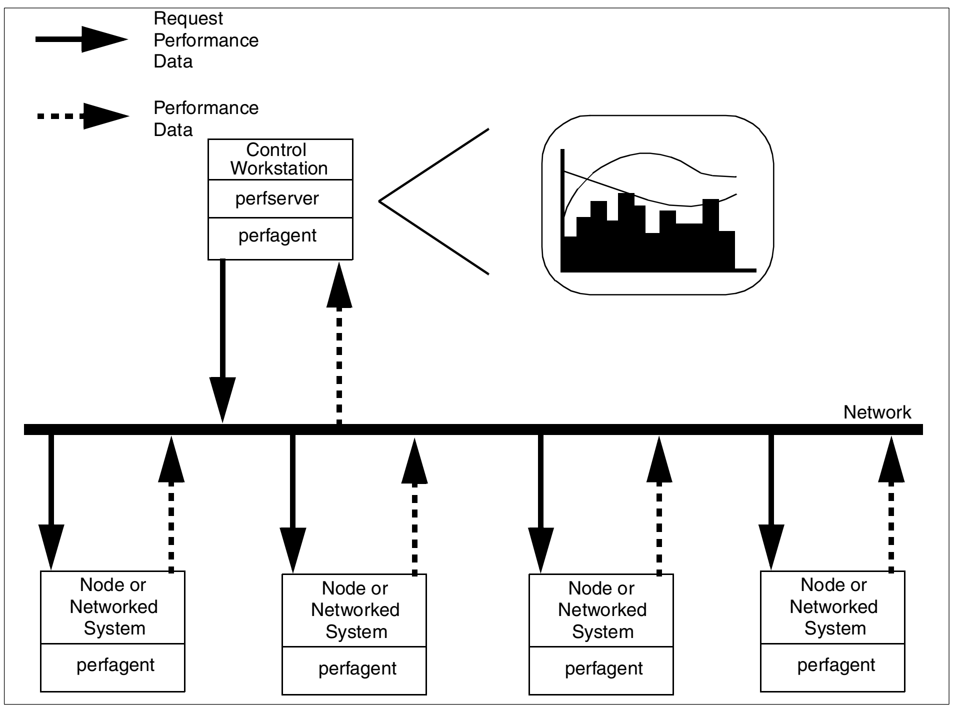

Figure 56. PTX/6000 network monitoring - client/server model

The ability to monitor local or remote systems and the network is important in an RS/6000 SP environment, where a large number of nodes need to be monitored using a single point of control.

PTX/6000 has the following features:

- Monitors system resources
- Analyzes system resource statistics
- Uses a Graphical User Interface (GUI)
- Is able to process Simple Network Management Protocol (SNMP) requests
- Supports the RS/6000 SP Performance Toolbox Parallel Extensions (PTPE)
- Has an Application Programming Interface (API) to create custom-written applications for analysis of performance archives

A.1.1 PTX/6000 installation

In an RS/6000 SP environment, we recommend that PTX/6000 be installed on the control workstation and any nodes that are to be monitored or managed.

We also recommend that the control workstation be used as a PTX/6000 server. The control workstation is designed to be a single point of management. Using a node introduces another point of management, and adds an additional system overhead.

A.1.1.1 Installation of PTX/6000 on the control workstation

The steps we used to install PTX/6000 on our control workstation were:

1. Create a PTX/6000 install directory:

   ```
   mkdir -p /spdata/sys1/install/ptx
   ```

2. Change the working directory to the install directory:

   ```
   cd /spdata/sys1/install/ptx
   ```

3. Load the PTX/6000 file sets into the install directory. The method used will vary depending on the location and media available, but use the `smitty bffcreate` fastpath.

4. Install the PTX/6000 performance management software:

   ```
   installp -aXd /spdata/sys1/install/ptx perfmgr
   ```

5. Install the PTX/6000 local monitoring software:

```
installp -aXd /spdata/sys1/install/ptx perfmgr.local
```

6. Install the PTX/6000 network monitoring software:

   ```
   installp -aXd /spdata/sys1/install/ptx perfmgr.network
   ```

7. Verify the installation:

   ```
   lslpp -l perfmgr.\*
   ```

8. The PTX/6000 agent software is a prerequisite. It should therefore already be installed.

   To check if it is installed, use `lslpp -l perfagent.*`.

   If it has not been installed, use the following steps to install the PTX/6000 agent software:

   a. Install the PTX/6000 remote agent:

      ```
      installp -aXd /spdata/sys1/install/ptx perfagent.server
      ```

   b. Install the PTX/6000 remote agent tools:

      ```
      installp -aXd /spdata/sys1/install/ptx perfagent.tools
      ```

   c. Verify the installation:

      ```
      lslpp -l perfagent.\*
      ```

9. Export the install directory from the control workstation to the nodes.

A.1.1.2 Installation of PTX/6000 on each node

To check if the perfagent software has already been installed, use `lslpp -l ‘perfagent.\*’`. If perfagent.server, or perfagent.tools are not installed, the following steps will install them:

1. NFS-mount the installation directory:

   ```
   mount <control workstation>:/spdata/sys1/install /mnt
   ```

2. Install the PTX/6000 remote agent:

   ```
   installp -aXd /mnt/ptx perfagent.server
   ```

3. Install the PTX/6000 remote agent tools:

   ```
   installp -aXd /mnt/ptx perfagent.tools
   ```

4. Verify the installation:

   ```
   lslpp -l ‘perfagent.\*’
   ```

A.2 Performance toolbox parallel extensions (PTPE)

PTPE is an extension of the Performance Toolbox (PTX/6000) for use in an RS/6000 SP complex.

PTPE is a scalable performance monitor, and when installed in an RS/6000 SP complex, it provides easy access to performance information.

Any node or group of nodes can be monitored with a single point of control. The performance parameters, and how the information is to be displayed, are definable. Monitoring can be in real time, or the performance data can be archived for later analysis.

PTPE performance statistics can be viewed as follows:

- On a per-node basis
- Averaged across a group of nodes
- Averaged across an RS/6000 SP complex

PTPE setup, configuration, and management has been integrated into Perspectives. PTPE also supports configuration and setup using AIX commands.

An API is provided. This allows development of customized applications to extract information from a PTPE archive.

A.2.1 PTPE installation

PTPE must be installed on the control workstation and on each node.

Note: Install PTPE on the control workstation *before* installing PTPE on the nodes.

The prerequisites before installing PTPE are:

1. AIX version 4 or later.
2. RS/6000 Cluster Technology.
3. Performance Aide for AIX must be installed on the control workstation and all nodes that will be monitored.
4. Performance Toolbox (PTX/6000) must be installed on any node or control workstation that will be used to display or analyze the performance data.

A.2.1.1 Installation of PTPE on the control workstation

The steps we used to install PTPE on the control workstation were:

1. Create a PSSP install directory:

   ```
   mkdir -p /spdata/sys1/install/pssplpp/PSSP-3.#
   ```

   The name of the PSSP directory will depend on the version of PSSP that is used; substitute # for the release number, such as 1 or 2.
2. Change the working directory to the PSSP install directory:

   ```
   cd /spdata/sys1/install/pssplpp/PSSP-3.#
   ```
3. Load the PSSP software into the PSSP install directory. The method used will vary depending on the location and media chosen. We used ftp to retrieve the PSSP software from an ftp server or use `smitty bffcreate` if the software is available in CD-ROM.
4. Install the PTPE programs:

   ```
   installp -aXd /spdata/sys1/install/pssplpp/PSSP-3.# ptpe.program
   ```
5. Install the PTPE documentation:

   ```
   installp -aXd /spdata/sys1/install/pssplpp/PSSP-3.# ptpe.docs
   ```
6. Verify the installation:

   ```
   lslpp -l pte \*
   ```
7. Register the PTPE spdmd daemon:

   ```
   /usr/lpp/ptpe/bin/spdmdctrl -a
   ```
8. Export the PSSP install directory from the control workstation to the nodes.
9. Create a PTPE monitoring group:

   ```
   /usr/lpp/ptpe/bin/ptpegroup
   ```

A.2.1.2 Installation of PTPE on each node

The steps we used to install PTPE on each node were:

1. NFS-mount the PSSP install directory:

   ```
   mount <control workstation>:/spdata/sys1/install/pssplpp /mnt
   ```
2. Install the PTPE programs:

   ```
   installp -aXd /mnt/PSSP-3.# ptpe.program
   ```
3. Verify the installation:

   ```
   lslpp -l pte \*
   ```
4. Add a supplier resource entry:

   ```
   cp /usr/samples/perfagent/server/xmservd.res /etc/perf
   ```

```
echo "supplier: /usr/lpp/ptpe/bin/ptpertm -p" >>
/etc/perf/xmservd.res
```

5. Register the PTPE spdmd daemon:

```
/usr/lpp/ptpe/bin/spdmdctrl -a
```

Appendix B. How to report performance problems to IBM

If you had no success using this book to solve possible performance problems, or to make your SP System perform as “fast” as possible, you still have a way to report the problem to IBM.

We will show you ways to report problems to IBM, and what you need to provide us to make the best progress in solving your problem. There are tools available to collect performance data. It is much easier than typing all the different commands step by step. Support personnel world wide will ask you for specific information, but here you have a common point to start with.

One useful tool that is provided by IBM is called *perfpmr*. You can simply download it as an anonymous user from the following ftp server:

Note

To use anonymous login, go to:

`ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr/`

and select the appropriate AIX Version for the perfpmr tool.

With the ftp tool of your choice (internet browser or ftp command) you can chose your AIX Level for the perfpmr tool and download it to your local machine. Follow the README instructions for further details.

The perfpmr tool is a collection of scripts used to gather performance data for IBM customer support personnel to use in resolving AIX system or application performance problems on RS/6000 systems. In addition to that tool, we suggest you provide a `snap -bgc` output for additional data (lslpp output and more). Please check you have enough file system space in /tmp before you start collecting data.

Before or after collecting the data you may contact your local IBM representative to open a Problem Record (PMR), which is the correct way to proceed with such problems. Refer to Table 48 for a quick checklist.

Table 48. Checklist

How to report performance problems to IBM - CHECKLIST
Collect data with the IBM provided perfpmr tool. Refer toTable 50 on page 478 for usage (use README for details).
Collect additional data with the `snap -bgc` command.

How to report performance problems to IBM - CHECKLIST

If not already done, CONTACT YOUR LOCAL IBM SUPPORT or Business Partner to open a problem record

If you have Internet access, please look at:

`http//ps.software.ibm.com/pbin-usa-ps/getobj.pl?/pdocs-usa/phonenos.html`

for a World Wide Contact list that includes phone numbers, mailing addresses and Internet addresses from A to Z.

You can also call the IBM International Assist Line for further details (see also in the Internet at:
`http://www.ibm.com/support/globalofferings/ial.html` and
`http://www.ibm.com/planetwide/`)

USA: 1-800-IBM-4-YOU or 404-238-1234.
EMEA: (44) 0 1475-898-134

Send the data to the responsible Support function.

For the USA, send it via ftp to:

`testcase.software.ibm.com`

Contact your local IBM support representative for the ftp directory location where you will store your test case file.

The mailing address is for USA Customers:

IBM Corp. / Zip 9551 / Bldg. 905
Attn.: AIX Testcase Dept. H8TS
11400 Burnet Road
Austin, Texas 78758-3493 USA

World Trade customers: please contact your local IBM representative for a ftp/mailing address.

Hopefully, we can provide you with a solution after providing the requested information. By using the above procedure, you can at least accelerate the process.

Quick overview of perfpmr usage

In addition to the regularly used README for the perfpmr script, we provide you with a short overview on how to run the program after you have successfully installed it on the node of your choice. Refer to Table 49 for installation details.

Table 49. Installation of perfpmr

INSTALLING THE PACKAGE

The following assumes the tar file is in /tmp and named 'perf433.tar.Z'.
 	
Login as root or use the 'su' command to obtain root authority
 	
Create perf433 directory and move to that directory (this example assumes the directory built is under /tmp)
 	
`# mkdir /tmp/perf433`
`# cd /tmp/perf433`
 	
Extract the shell scripts out of the compressed tar file:
 	
`# zcat /tmp/perf433.tar.Z | tar -xvf -`
 	
Install the shell scripts
 	
`# sh ./Install`

Now we are able run the perfpmr script simply by typing it. To gather more information, we also issue the `snap -bgc` command, which will help you gather general information about the system including `lslpp`, ODM data, and much more. See Table 50 on page 478 for more details.

Table 50. Usage of perfpmr

Usage of perfpmr and snap -bgc
Check for enough file system space in the /tmp directory or the directory you chose. Thumbvalue: 12 MB file system space per processor (check with `df -k` command) Create a working directory: # `mkdir /tmp/perfdata` # `cd /tmp/perfdata` Attention HACMP users: It is generally recommend HACMP deadman switch interval be lengthened while performance data is being collected. Collect our 'standard' PERF433 data for 600 seconds (600 seconds=10 minutes) # `perfpmr.sh 600` Answer the questions in the provided file "PROBLEM.INFO" for general info Now we need to tar the data into one compressed file: # `cd /tmp/perfdata` (or whatever directory used to collect the data) # `cd ..` # `tar -cvf perfdata.tar perfdata` # `compress perfdata.tar` The resulting file will be named 'perfdata.tar.Z'. Now we need the snap data: # `snap -bgc` This command automatically creates a subdirectory in /tmp called /tmp/ibmsupt and checks for file system space requirements. It also automatically creates a file named snap.tar.Z in the /tmp/ibmsupt subdirectory. You are now able to combine both files perfdata.tar.Z and snap.tar.Z in one file that will be sent to IBM Support.

Appendix C. Hardware details

In this appendix, we give an overview of the nodes and attached servers available at the time the book was written, plus a node selection criteria scheme that may help you find the right node for your application.

C.1 Node types

The SP processor nodes available for the SP are in most cases equivalent to specific RS/6000 machines. The most recent nodes are all PCI-based machines. and there are no more MCA nodes like the 135 MHz wide node available (they have been withdrawn from marketing but there are still some types used in current configurations). To give you an overview of some of the different node types, refer to Table 51 on page 479

Table 51. SP nodes overview

Node Type	Processor	Data Cache	MCA/PCI Slots Available		Memory Bus Bandwidth
			no Switch	using Switch	
Thin Node 160 MHz (397)	P2SC	128 KB	4	3	256-bit
Wide Node 135 MHz (595)	P2SC	128 BK	7	6	256-bit
High Node 200 MHz (J50)	PowerPC [1]	32K + 2 MB L2 [5]	14	13	256-bit
Thin Node 332 MHz (H50)	PowerPC [2]	32K + 256 KB L2 [5]	2 [6]	2 [7]	128-bit
Wide Node 332 MHz (H50)	PowerPC [2]	32K + 256 KB L2 [5]	10 [6]	10 [7]	128-bit
S70 Node 125 MHz	PowerPC [3]	64K + 4 MB L2 [5]	11 [6]	8 [8]	Dual 512-bit
S7A Node 262 MHz	PowerPC [4]	64K + 8 MB L2 [5]	11 [6]	8 [8]	Dual 512-bit
S80 Node 450 MHz	PowerPC [11]	512K + 8 MB L2 [5]	11 [6]	8 [8]	Quad 512-bit
Thin Node 200 MHz (260)	POWER3 [9]	64K + 4 MB L2 [5]	2 [6]	2 [7]	128-bit
Wide Node 200 MHz (260)	POWER3 [9]	64K + 4 MB L2 [5]	10 [6]	10 [7]	128-bit
Thin Node 375 MHz (270)	POWER3-II [12]	64K + 8 MB L2 [5]	2 [6]	2 [7]	128-bit
Wide Node 375 MHz (270)	POWER3-II [12]	64K + 8 MB L2 [5]	10 [6]	10 [7]	128-bit
High Node 222MHz	POWER3 [13]	64K + 4 MB L2 [5]	5 [10]	5 [10]	2048_bit
High Node 375MHz	POWER3-II [14]	64K + 8 MB L2 [5]	5 [10]	5 [10]	2048-bit

1. 2/4/6/8-way PowerPC 604e.
2. 2/4-way PowerPC 604+LX.
3. 4-way PowerPC RS64 64-bit.
4. 4-way PowerPC RS64 II 64-bit.
5. This data cache is for each pair of processors.
6. These slots are PCIs.
7. SP Switch MX Adapter does not use an I/O slot.
8. SP System Attachment Adapter requires 3 PCI slots (Adapter into Slot 10, slots 9 and 11 remain empty).
9. 1/2-way 64-bit processor.
10. RIO expansion Box offers 8 PCI Slots, a maximum of 6 RIO boxes equals 53 total PCI Slots available (CPU + RIO)
11. 6-way PowerPC RS64 III 64-bit
12. 2/4-way POWER3-II 630+ 64-bit
13. 2/4/6/8-way POWER3 630FP 64-bit
14. 4/8/12/16-way POWER3-II 64-bit

Characteristics of these nodes vary by:

- The bandwidth of the internal bus
- The number of CPUs (the high nodes are SMP systems with 2, 4, 6, or 8 processors)
- The maximum amount of memory
- The number of available micro channels or PCI adapter slots
- The amount of memory bandwidth (one word = 32 bits)

The memory bandwidth is important when considering system performance. The memory bandwidth word size specifies how many words of data can be moved between memory and data cache per CPU cycle.

C.2 Roles of nodes

There are a variety of nodes to choose from when you configure an SP system. The following is a short overview that gives some guidance for your node selection.

The latest 375 MHz POWER3 SMP high node has over three times the performance of it’s predecessor, the 222MHz POWER3 SMP high node. The capability to have up to 6 external I/O Drawers (RIO) gives you a huge number of PCI slots available (53 PCI slots). The upcoming SP Switch2, that is only available for POWER3 SMP high nodes, offers you more than three times the bandwidth of the SP Switch for interconnecting POWER3 SMP high nodes.

Thin node
Thin nodes are suitable for a wide range of applications, including all commercial applications, and most scientific and technical applications.

Wide node
Wide nodes in some cases demonstrate superior performance over thin nodes even though they share the same type and number of processors. This is due to the existence of a second controller, in the wide node, that attaches eight additional slots to the internal bus. For some I/O-related performance this can be significant. For example, with SSA disks, the data rate performance increases from about 40 MB/second on a thin PCI node to more like 70 MB/second on a wide PCI node. This alone might be reason enough for selecting wide nodes rather than thin nodes. The incremental cost in using wide nodes rather than thin nodes may well be a good investment if balanced performance is of importance. A specific application example where the extra performance is likely to be observed when using a PCI wide node would be when using the PCI node as a disk server node for a GPFS file system. Up to double the bandwidth could be achieved with a wide node in this case.

High node
High nodes are more suited for multi threaded applications, such as commercial database processing, that require significant number of PCI adapters. The POWER3 SMP high nodes support the SP Switch with the SP Switch MX2 adapter and the new SP Switch2 with the SP Switch2 adapter.

S70/S7A/S80 SP attached server
The S70/S7A/S80 Attached Server is a PowerPC RS/6000 model capable of 32- or 64-bit processing. It is best suited to be a powerful database server. The S70/S7A nodes can have 4-, 8- or 12 processors and up to 32 GB of memory. The S70 has a 125 MHz PowerPC RS64 processor. The S7A has a PowerPC RS64II processor with 262 MHz. The S80 has a 450MHz PowerPC RS64III processor and can have 6-, 12-, 18- or 24 processors and up to 64 GB of memory

Router node
Router nodes are dependent nodes that extend the RS/6000 SP system's capabilities. They are not housed in the frame but are dependent on the switch existing in the environment. An RS/6000 SP Switch Router is a high-performance I/O gateway for the RS/6000 SP system and provides the fastest available means of communication between the RS/6000 SP system and the outside world, or among multiple RS/6000 SP systems. The Ascend

GRF switched IP router can be connected to the SP switch via the SP Router Adapter. The Switch Router Adapter provides a high-performance 100 MB/sec full duplex interface between the SP Switch and the Ascend GRF. The GRF (IBM Machine Type 9077) is not supported on the new SP Switch2 that is coming, it is restricted on just the SP Switch.

Codenames for the nodes

Do you wonder perhaps, when a Support Person asked you something about your node? For example, you were already asked “Do you have a Silver node?” and you were not able to answer that? During the design of new hardware, this new hardware gets a Code Name. Here is a short overview of actual code names for nodes that are used in SP Systems:

- Silver Node is a 332 MHz SMP Thin/Wide node.
- Winterhawk is a POWER3 SMP 200 MHz Thin/Wide node.
- Winterhawk2 is a POWER3 SMP 375 MHz Thin/Wide node.
- Nighthawk is a 222 MHz SMP High node.
- Nighthawk2 is a 375 MHz SMP High node.

If you have more questions regarding those kinds of questions, do not hesitate to ask your Service representative.

C.3 Communication paths

Two communication paths between the nodes and the Control Workstation (Ethernet network) and between the frame and the Control Workstation are mandatory for an SP system. The switch network is optional.

RS232 Hardware monitoring line

The mandatory RS232 hardware monitoring line connects the CWS to each RS/6000 SP frame, used primarily for node and frame hardware monitoring.

Ethernet

One of the prerequisites of the RS/6000 SP is an internal bnc or 10BaseT network. The purpose of this mandatory network is to install the nodes’ operating systems and PSSP software, as well as to diagnose and maintain the RS/6000 SP complex through the PSSP software.

C.4 System partitioning

System partitioning within the RS/6000 SP allows you to divide your system into logically separate systems. The concept of system partitioning is very similar to the Virtual Machine in the mainframe environment, which supports different machine "images" running in parallel; you can have a production image and test image within the same machine. The RS/6000 SP provides completely isolated environments within the complex.

Although RS/6000 SP system partitioning has been used as a migration tool to isolate test environments, this is not its primary purpose. Running various versions of AIX and PSSP is possible without having to partition.

Now that you can differentiate between production and testing within a complex, you can simulate a production environment and take all necessary steps to tune your system before migrating to the true production environment.

C.5 Node selection process

There are different criteria for choosing the right node for your application. One criterion may be capacity; how many adapters, how many internal disks, or how much memory will fit into it.

Capacity

When selecting a node, you may choose to use the methodology shown in the flowchart in Figure 57 on page 484. Look at the required capacity for adapters, disks, and memory -- this will help you decide whether a thin node has sufficient capacity. As you consider this area, take particular care to include all adapters that will be required, both now and in the near future.

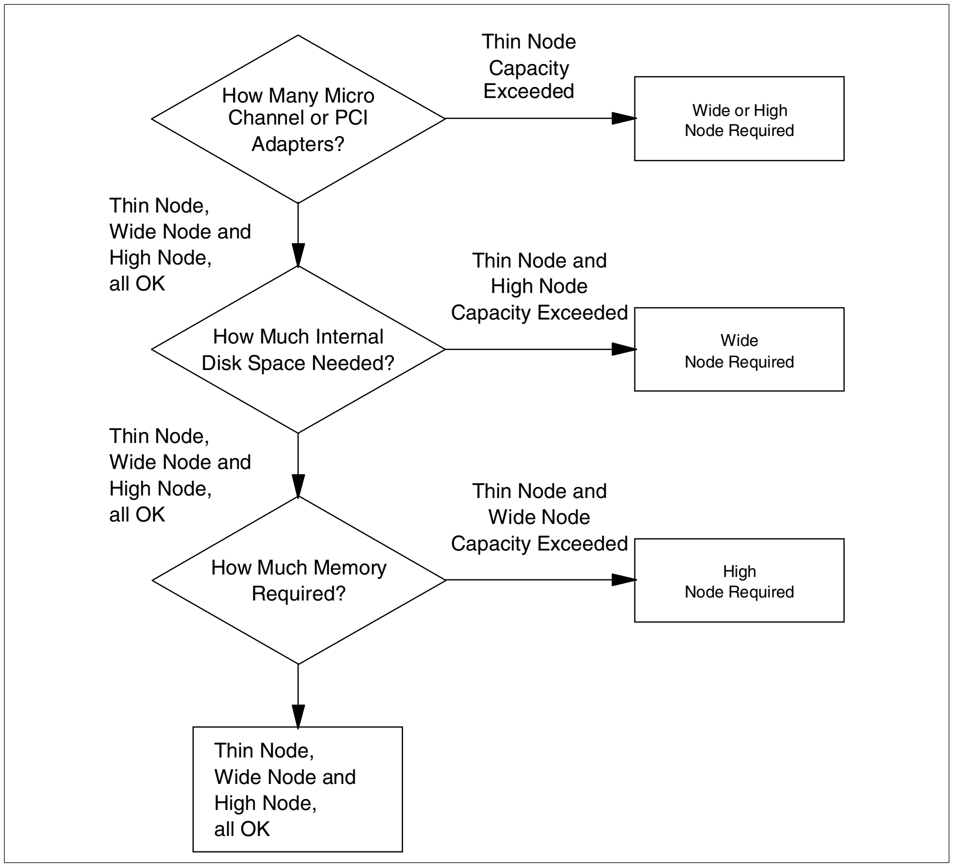

Figure 57. RS/6000 SP node selection based on capacity

Some of the selections that need to be made are quite straightforward decisions and are based on concrete factors, such as the number of adapters needed in this node, or the amount of memory required.

Performance

The flow chart in Figure 58 on page 485 can be used to help make the right choice of nodes to select from a performance perspective. It assumes you have already made a decision on the basis of capacity as discussed earlier.

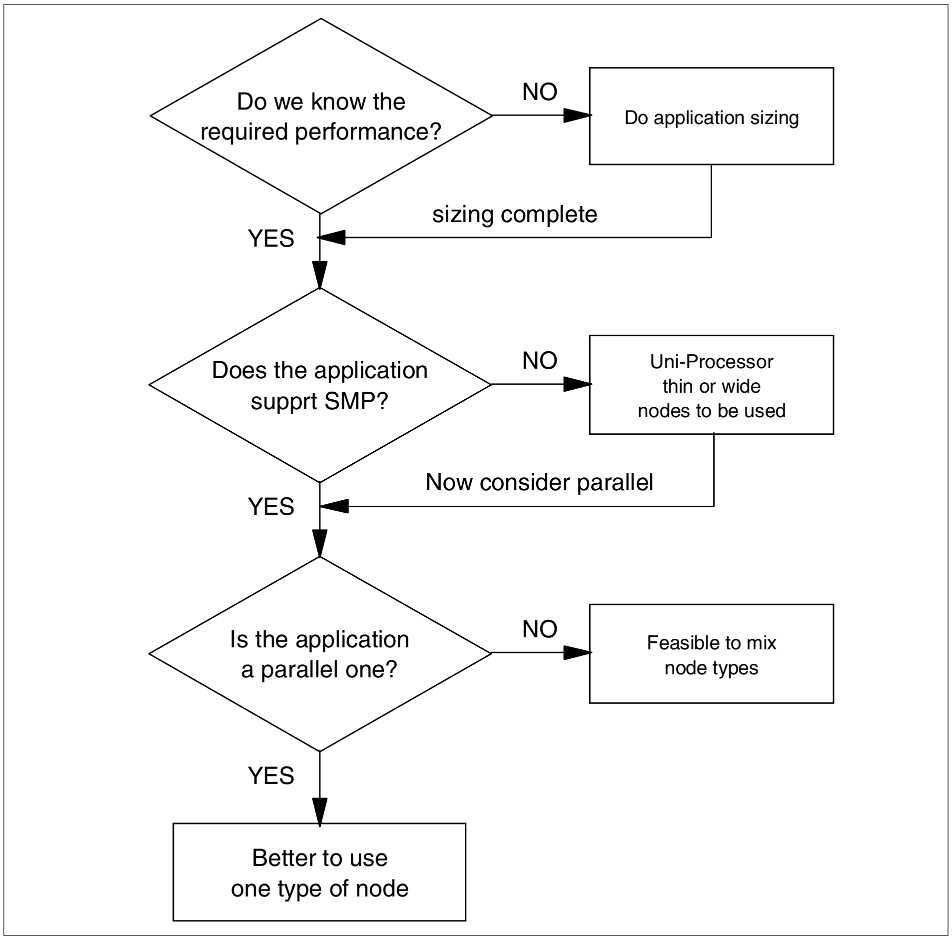

Figure 58. RS/6000 SP node selection based on performance

Performance measurements

It is important to select the correct nodes for the applications that we wish to implement and that exploit the most appropriate architecture: serial or parallel, uniproccesors or SMP processors. Make sure that you have a cost-effective solution that also allows easy upgrades and a growth path as required.

It is important not to be swayed in these considerations by any price/performance considerations that force you to make compromises with regard to the ideal nodes for a specific purpose. The good news is that price/performance for all RS/6000 SP nodes, uniprocessor or high nodes, is similar for the current selection of nodes.

Appendix D. Special notices

This publication is intended to help RS/6000 SP users, system administrators, and systems engineers understand the available performance monitoring and system tuning tools on RS/6000 SP and to undertake a detailed performance analysis. The information in this publication is not intended as the specification of any programming interfaces that are provided by RS/6000 SP or POWERparallel System Support Programs. See the PUBLICATIONS section of the IBM Programming Announcement for RS/6000 SP and POWERparallel System Support Programs for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AS/400
IBM ®
Lotus Notes
PAL
PowerPC 604
Redbooks Logo
SP
WebSphere
Domino
Lotus
OS/400
POWERParallel
Redbooks
RS/6000
System/390

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere.,The Power To Manage., Anything. Anywhere.,TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

WebStone is a freeware product owned by Mindcraft, Inc.

Apache Bench and JMeter are owned by the Apache Software Foundation

Rational Suite Performance Studio is owned by the Rational Software Corporation

Webload is owned by RadView Software, Inc

LoadRunner is owned by Mercury Interactive Software

Other company, product, and service names may be trademarks or service marks of others.

Appendix E. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

E.1 IBM Redbooks

For information on ordering these publications see “How to get IBM Redbooks” on page 495.

- *Database Performance on AIX in DB2 UDB and Oracle Environments*, SG24-5511
- *GPFS: A Parallel File System*, SG24-5165
- *IBM eNetwork Communcations Server for AIX: Understanding and Migrating to Version 5: Part 2 - Performance*, SG24-2136
- *Oracle8i Parallel Server on IBM SP Systems: Implementation Guide*, SG24-5591
- *RS/6000 Performance Tools in Focus*, SG24-4989
- *Sizing and Tuning for GPFS*, SG24-5610
- *Understanding IBM RS/6000 Performance and Sizing*, SG24-4810
- *Websphere V3 Performance Tuning Guide*, SG24-5657

E.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at ibm.com/redbooks for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
IBM System/390 Redbooks Collection	SK2T-2177
IBM Networking Redbooks Collection	SK2T-6022
IBM Transaction Processing and Data Management Redbooks Collection	SK2T-8038
IBM Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
IBM AS/400 Redbooks Collection	SK2T-2849
IBM Netfinity Hardware and Software Redbooks Collection	SK2T-8046
IBM RS/6000 Redbooks Collection	SK2T-8043
IBM Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

E.3 Other resources

These publications are also relevant as further information sources:

- *AIX Command Reference*, GBOF-1802
- *AIX System Management Guide: Communications and Networks*, SC23-4127
- *AIX Technical Reference Volume 1*, SN32-9029
- *Communications Server for AIX Planning and Performance Guide*, SC31-8220
- *DB2 UDB Administration Guide: Performance V6*, SC09-2840
- *DB2 UDB System Monitor Guide and Reference V5*, S10J-8164
- *IBM PSSP Programs: Managing Shared Disks*, SA22-7439
- *IBM General Parallel File System for AIX: Administration and Programming Reference*, SA22-7452
- *IBM General Parallel File System for AIX: Concepts, Planning, and Installation,* GA22-7453
- *IBM General Parallel File System for AIX: Data Management API Guide*, GA22-7435
- *IBM General Parallel File System for AIX: Installation and Tuning Guide,* GA22-7453
- *IBM General Parallel File System for AIX: Problem Determination Guide*, GA22-7434
- *IBM DB2 UDB Administration Guide: Performance*, SC09-2840
- *PSSP Administration Guide*, SA22-7348
- *PSSP: Managing Shared Disks*, SA22-7279
- *RS/6000 SP: Planning, Volume 1, Hardware and Physical Environment*, GA22-7280
- *Websphere Application Server Enterprise Edition V3.0,* G325-3920

E.4 Referenced Web sites

These Web sites are also relevant as further information sources:

- `http://www.tivoli.com/products/solutions/storage` - General information about Tivoli storage products

- `http://www.rs6000.ibm.com/support/sp` - Main site for RS/6000 SP information and support
- `http://www.rs6000.ibm.com/support/sp/perf/` - RS/6000 SP Support Tuning Information Page

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site ibm.com/redbooks**

  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States or Canada	pubscan@us.ibm.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

Title	Order Number	Quantity

First name | Last name

Company

Address

City | Postal code | Country

Telephone number | Telefax number | VAT number

☐ Invoice to customer number ______________

☐ Credit card number ______________

Credit card expiration date | Card issued to | Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Index

Symbols

/dev/kmem 325
/dev/lv# 155
/dev/mem 246
/etc/environment 293
/etc/exports 69
/etc/filesystems 79, 224
/etc/hosts 111, 181, 238, 293
/etc/netsvc.conf 111, 181, 238
/etc/rc.net 41, 47, 182
/etc/rc.nfs 81
/etc/rc.sp 84
/etc/resolv.conf 238, 293
/etc/security/limits 204, 212, 309
/etc/trcfmt 322
/etc/xtab 69
/tftpboot 9, 84
/tftpboot/script.cust 13
/tftpboot/tuning.cust 9, 13, 84, 182
/tmp 254
/tmp/log 297
/unix 246
/usr 228, 230
/usr/lpp/ssp/css/estat 233
/usr/lpp/ssp/install/config 85
/usr/lpp/ssp/install/tuning.default 85
/usr/samples/kernel/schedtune 18, 137
/usr/samples/kernel/vmtune 18, 154
/usr/sbin/acct/accton 243
/usr/sbin/perf/diag_tool 298
/usr/sbin/perf/diag_tool/pdt_config 296
/usr/sbin/perf/diag_tool/pdt_report # 299
/usr/tivoli/tsm/client/ba/bin/dsm.sys 345
/usr/tivoli/tsm/server/bin/dsmserv.opt 339
/var/mmfs/etc/cluster.preferences 389
/var/perf/cfg/diag_tool/.collection.control 298
/var/perf/cfg/diag_tool/.files 300
/var/perf/cfg/diag_tool/.nodes 300
/var/perf/cfg/diag_tool/.reporting.list 298
/var/perf/cfg/diag_tool/.thresholds 300
/var/perf/tmp 298
/var/perf/tmp/PDT_REPORT 299
/var/samples/perfagent/server 319

Numerics

3dmon 217

A

accounting filesystem 243
acctcom 243
accton 243
accumulated CPU time (TIME) 219
ActiveX 397
Adapter driver 76
adapter queue 88, 90
Adapter queue overflows 201
Address Resolution Protocol 40
Address Resolution Protocol (ARP) 193
AIX commands
  lsattr 406
  no 406
  vmtune 449
AIX network tunables 45, 190
AIX parameters
  maxperm 447
  minperm 447
AIX performance tools
  lscfg 406
  nmon 410
  vmstat 409
  vmtune 449
AIX tunables 46
AIX V4.3.3 132
AKtools 397
allocation methods for working storage 127
Apache Bench 397
Apache Software Foundation 397
application
  data mining 90
application programming interface 470, 472
application spinner 221
application type
  commercial 481
  commercial database 481
  technical 481
application_program command 140
ARP 40, 238
arp 41, 240
  calculating entries 40
  calculation 40
ARP cache 190

ARP cache sizing 41
ARP cache thrashing 240
ARP cache-related tunables
  arpqsize 59
  arpt_killc 59
  arptab_bsiz 58
  arptab_nb 58
  ifsize 59
arpqsize 40, 41, 59
arpt_killc 59
arptab_bsiz 40, 58
arptab_bsize 84
arptab_nb 40, 41, 58, 84, 193
Ascend 481
Asynchronous I/O 442
ATM 29, 44
ATM tuning 112
atmstat 96
autorestart 18

B
back end processes 434
back-file system 78
Backup 412
bad-block relocation 156
Balanced systems 411
Balancing workloads 412
basic adapter settings 15
  rx_que_size 16
  rxbuf_pool_size 16
  tx_que_size 16
basic AIX system settings 18
basic architecture 21
basic network settings 14
basic performance settings 13
Batch workload 450
benchmarking 7
bf (bigfoot) 244
bfrpt 244
biod 64
biod daemon 69
biod daemons 72
block size 370
bosboot 205
bottleneck 142
bottlenecks 22
Bottlenecks on the I/O bus or disks 201
buddy buffer wait_queue size 378
Buddy buffers sizing 201
buffer pools 29

C
Cache File System 78
cache option 357
caches 15
calculating the priority value for a process 133
center 175
changing the AIX network tunables 9
chdev 16, 19, 50, 80, 145, 151, 183
chfs 166
chgcss 99
chgcss parameter
  rpoolsize 196, 354
  spoolsize 196, 354
chlv 159
chnfs 73
Choosing minfree and maxfree settings 119
Choosing minperm and maxperm settings 124
chps 167
chvg 159
client server 469
client storage 117
client-polling 341
clr_q attribute 145
cluster technology 472
clustered system 402
col_dump 203, 247
collision detection 250
command
  acctcom 243
  accton 243
  arp 41, 240
  arp -a 231
  atmstat 96, 284
  awk 252
  bosboot 205
  chdev 16, 19, 50, 80, 98, 145, 151, 183
  chfs 166
  chgcss 99, 100, 102
  chlv 159
  chnfs 73
  chps 167
  chvg 159
  col_dump 247
  cp 167
  crash 218, 246

createvsd 357
cronadm 297
defragfs 179
dsh 189
entstat 96, 248, 284
errpt 81, 407
estat 251
exportfs 69
fddistat 96, 284
fencevg 364
filemon 228, 251
fileplace 256
fsck 165, 166
getlvcb 268
ha.vsd 357
hatstune 258
host 237, 293
ifcl_dump 260
ifconfig 29, 32, 55, 231
iostat 73, 216, 222, 260
ipcs 446
iptrace 192, 238, 262
lsattr 14, 15, 98, 100, 149, 152, 205, 232, 406
lslv 158, 223, 266
lsps 167, 265
lspv 223, 266
lsvg 223, 267
lsvsd 357
migratepv 159, 164
mklv 164, 445
mmchconfig 383, 384
mmcrfs 381
mmdf 270
mmfsadm 270
mmlsdisk 270
mmlsfs 270, 382
mmlsmgr 269, 388
mount 71, 291
ndb 231, 271
netpmon 240, 275
netstat 25, 27, 28, 44, 73, 76, 99, 196, 197, 231, 232, 281, 286
netstat -i 81
netstat -s 28
nfso 9, 45, 63
nfso -a 231
nfsstat 287
nfsstat -cr 80
nice 8, 133
no 9, 27, 40, 45, 46, 182, 191, 285, 406
  route_expire 34
  subnetsarelocal 35
no -a 231
nslookup 237, 293
ntpq 295
ntptrace 295
ping 263, 302
pprof 303
ps 290, 304, 446
pstat 213, 214, 218, 308
renice 8, 133
REORG 420
reorgvg 159, 177
rmdev 79
sar 139, 207, 215, 309, 450
  block I/O 311
  context switching 312
  processor utilization 310
  run queue 311
  system calls 312
schedtune 8, 18, 130, 137
schedtune -t 138
sleep 244
spmon 313
ssaconn 147
startsrc -s 79
statvsd 313, 361
stopsrc -s 79
svmon 120, 125, 207, 314
svmon -dlC 208
svmon -dU 211
syncvg 159
tar 168
tcpdump 238, 317
  ARP 318
  TCP 318
  timestamps 318
tokstat 96, 248, 284
topas 217, 319
tprof 320
  processes 320
  shared libraries 320
trace 322
traceroute 321
trcrpt 324
umount 165
unfencevg 364
updatevsdnode 358

updatevsdtab 357
varyoffvg 364
varyonvg 364
vdidl3 108, 355
vdidl3xx 108
vmstat 73, 130, 205, 207, 213, 324, 409
column descriptions 326
examples of usage 329
vmtune 9, 18, 120, 153, 154, 205, 330, 446
numperm 331
vsdatalst 360
xmperf 210, 331
Command tag queuing for non-IBM SCSI disks 145
commands
lscfg 406
vmtune 406
communication protocols for the SP Switch adapter 101
communications memory buffers 463
Computational memory or computational pages 118
Concurrent VSD 363
Connection oriented 291
Connectionless 291
consolidated system 22
cookies 397
CPU 129, 189
RT_GRQ variable 130
cpu 212
CPU monitoring tools 203
CPU tuning 408
CPU utilization (CP) 219
cpu_hard 212
crash 218, 246, 271
cronadm 297
CVSD implementation 364
CWS
ipforwarding 182
sb_max 182
tcp_mssdflt 182
tcp_recvspace 182
tcp_sendspace 182
thewall 182
udp_sendspace 182
upd_recvspace 182
CWS default route 22
cylinder 143

D

damage /etc/rc.net 84
damage /tftpboot/tuning.cust 84
Data
Placement policy 411
data 204
data/metadata 387
data_hard 204
database server 394
DB2 UDB
DB2MEMDISCLAIM 418
DB2MEMMAXFREE 418
Error log (db2diag.log) 416
Prefetchers 414
DB2 UDB commands
REORG 420
REORGCHK 420
DB2 UDB EEE
Fast Communication Manager (FCM) 403
Function shipping 403
Inter-partition parallelism 403
Intra-partition parallelism 403
DB2 UDB parameters
buffpage 413
catalogcache_sz 417
chngpgs_thresh 415
DB2_PARALLEL_IO 419
DB2_STRIPED_CONTAINERS 419
dbheap 417
intra_parallel 416
locklist 418
logbufsz 417
maxagents 417
maxappls 417
num_iocleaners 414
num_ioservers 414
pckcachesz 416
sheapthres 416
sortheap 415
stmtheap 416
DB2 UDB processes
db2agent 417
deadlocks 119
Dedicated disk 166
Deferred allocation 127
Deferred allocation algorithm 128
Deferred Page Space Allocation 166, 226
defrags 179
demux layer 28

dependent node 481
Depth of receive queue 464
detailed host information 280
detailed interface information 279
Detailed protocol 278
Detecting bad blocks 156
device driver buffers 15
device driver transmit statistics 277, 278
Device drivers 232
Disk
  Dedicated to particular tasks 411
  I/O 410
Disk I/O pacing 167
disk I/O statistics 230
Disk mirroring 157
Disk placement within a loop 146
DISK_STORAGE_BALANCE 301
Distributed Lock Manager (DLM) 433, 439
dma_bus_mem 152
dma_mem setting 151
DNS 237
document your system 6
dog threads 28, 32
  guidelines 32
double-buffering 126
dropped packets 75, 235
dsh 189
DSS 407, 450
dump options
  cfgmgr 271
  disk 271
  malloc 271
  pgalloc 271
  stripe 271
  tscomm 270
dynamic mtu discovery 283
dynamic tunables 9
dynamically changeable attributes 102

E

Early allocation 127
Early allocation algorithm 128
ECHO_REPLY 263
ECHO_REQUEST 263
edge 18
edge inter-disk allocation policy 18
Enabling KLAPI 367
Enhanced RAID 0 162
entstat 96, 248, 272
environment
  Commercial Database 86
  consolidated 22
  homogeneous 22
  parallel 111
  parallel database 90
  Scientific and Technical 86
  server
    typical 91
  Server Configuration 86
  Software Development 86
ERP 397
errpt 81
Escon 29
  DATABUFFERSIZE 113
  Interface MTU 113
Escon tuning 113
estat 251
Ethernet 29, 44, 191, 240
Ethernet network
  collisions 249
Ethernet routing 22
Ethernet values 16
EVENT_HORIZON 302
Excessive aggregate memory requirements 201
exclusive 209
Exhaustion of switch pools 201
exportfs 69
extendednetstats 48, 196, 233
external server considerations 195
extracting logical volume information 268

F

Fast-Write Cache 147
FCS 44
FDDI 29, 44, 191
FDDI tuning 112
fddistat 96
File
  /etc/hosts 111
file fragmentation 230
File System manager location 388
file transfer applications 466
filemon 228, 251
filemon command detailed output 255
filemon options
  all 252

lv 252
pv 252
vm 252
Files
$DIAGPATH/db2diag.log 407
/dev/lv# 155
/dev/mem 246
/etc/environment 293
/etc/exports 69
/etc/hosts 181, 293
/etc/netsvc.conf 111, 181, 238
/etc/rc.net 41, 47, 182
/etc/rc.nfs 81
/etc/rc.sp 84
/etc/resolv.conf 238, 293
/etc/security/limits 204, 212, 309
/etc/trcfmt 322
/etc/xtab 69
/tftpboot/tuning.cust 9, 13, 84, 182
/tmp/log 297
/unix 246
/usr/tivoli/tsm/client/ba/bin/dsm.sys 345
/usr/tivoli/tsm/server/bin/dsmserv.opt 339
/var/mmfs/etc/cluster.preferences 389
bos.adt 18
bos.adt.samples 330
db2diag.log 416
dsm.sys 340, 352
dsmserv.opt 352
fma.data 230
ipreport.out 253
iptrace.out 253
notes.ini 460
nsi.all 233
pprof.cpu 303
pprof.famcpu 304
pprof.famind 304
pprof.namecpu 304
pprof.start 303
tftpboot/script.cust 13
tuning.cust 6, 22
fixed-priority thread 133
fixed-priority threads 138
fragments dropped after timeout 28
Frequency 184
frequency 259
front end processes 434
front-file system 78
FS_UTIL_LIMIT 301
fsck 165, 166
full-duplex ports 146
Function shipping 403

G

getlvcb 268
Gigabit Ethernet 29, 161
Gigabit Ethernet tuning 111
GPFS
LAPI 386
network tuning
ipqmaxlen 373
SSA adapter tuning
max_coalesce 375
queue_depth 376
Switch adapter tuning
spoolsize/rpoolsize 374
VSD tuning
1 VSD per 1 hdisk 379
Concurrent VSD (CVSD) 379
Dedicated VSD server 380
VSD usage of KLAPI 379
VSD_max_buddy_buffer_size 377
VSD_max_buddy_buffers 378
VSD_maxIPmsgsz 379
VSD_pbufs 377
VSD_request_blocks 377
GPFs
VSD tuning
others VSD considerations 379
GPFS 1.3 269
GPFS file system integrity 371
GPFS overview 369
GPFS replication 371
GPFS tuning
comm_protocol 386
File system block size 380
maxFilesToCache 383
maxStatCache 384
Number of GPFS nodes 380
pagepool 382
prefetchThread 385
worker1Thread 385
worker2Thread 385
worker3Thread 385
graphical user interface 470
GUI
see graphical user interface

Gung ho reclamation 121

H
hardlimit 204, 212
hardware co-existence 483
hatstune 258
head 143
Header types
 ICMP 264
 IPX (PC protocol) 264
 NFS (RPC) 264
 TCP 264
 UDP 264
high node 481
HiPPI 29, 44
HIPPI tuning 112
HiPS 29
homogeneous system 22
host 237, 293
Hosts 300
How to set minfree and maxfree 120
How to set minperm and maxperm 125
HTTP 1.0 397
HTTP server 394

I
I/O 190
I/O errors 156
I/O monitoring tools 203
I/O tuning 409
I/O wait 409
ICMP statistics 236
ifcl_dump 260
ifconfig 29, 32, 55
ifsize 59, 84
IN BAND percentage 225
initial considerations 21
initial settings
 alternate tuning file 85
 commercial 85
 development 85
 scientific & technical 85
 typical server 85
initial tuning parameters 6
inner edge 175, 225
inner middle 175
input/output
 pacing disable 169
installation of a node 85
inter-disk 174
inter-disk Tuning considerations 175
Internet Control Message Protocol (ICMP) 302
Internet Network layer 23
Internet Protocol (IP) 23
Internet Transport Layer 23
intra-disk 174
Intra-partition parallelism 403
iodone 174
iostat 73, 216, 222, 260
IP receive flow 27
IP send flow 27
IP tunables inheritance rules 45
ipfilter 264
ipforwarding 182
ipfragttl time 28
ipintrq overflows 27
ipqintrq 287
ipqmaxlen 14, 27, 57
ipreport 203, 262
iptrace 192, 238, 262
IPv4 45
IPv6 45

J
JFS log 175
JMeter 397
Journaled File System (JFS) 164
JVM 394

K
keep-alive 397
Kernel mode 136
kproc 189
kprocprio 173
ksh 209

L
Late allocation 127
Late allocation algorithm 128
Latent demand 408
limitations of the renice command 139
LoadRunner 397
local file system 78
Logical resource access tuning 410
Logical resources 412

logical resources 8
Logical Volume Device Driver (LVDD) 155
Logical Volume Manager (LVM) 141
Logical Volume placement 445
Lotus Notes
Database Instance Identifier 458
Favor Restart Recovery Time 458
Favor Runtime 458
flushing 458
load balancing parameter
Server_Availability_Threshold 461
Server_MaxUsers 461
performance improvement 457
performance problems
CPU bottlenecks 461
displaying large views 460
NSF_BUFFER_POOL_SIZE 460
NSF_DBCACHE_MAXENTRIES 460
slow database access 460
performance tuning parameter
UPDATE_SUPPRESSION_TIME 461
Standard (default) 458
Transactional Logging 456
Transactional Logs 457
tuning
AMgr 460
Calconn 460
Collector Reporter 460
Router 460
tuning for performance 459
lrud kernel process 119
lsattr 14, 15, 98, 100, 149, 152, 205
lslv 158, 223, 225, 227, 266
lsps 167, 226, 265
lspv 223, 224, 266
lsvg 223, 267
LVM reading of data from striped logical volumes 162
LVM writing of data from striped logical volumes 162
lvm_bufcnt parameter 163

M

MAC
address translation 193
Massively Parallel Processors (MPP) 402
Max Packets 249, 284
max_coalesce 149
max_coalesce setting 148
Max_ttl value 322
maxfree 424
maximum segment size 33
local network 33
remote network 33
maximum transmission unit 28
maxmbuf 31, 47, 463
maxpageahead 121
maxperm 123, 424
maxperm parameter 71
maxpgahead 153, 424
maxpout 168
maxrandwrt 155
maxreqs 172
maxservers 173
maxuproc 18
mbufs 29, 196, 233
Memory
Tuning 408
memory 115
insufficient 115
memory bandwidth 480
memory management 18, 29
Memory monitoring 213
Memory monitoring tools 203
memory pools 30
Memory related tunables 46
thewall 46
extendednetstats 48
maxmbuf 47
rfc1323 52
sb_max 47
tcp_recvspace 50
tcp_sendspace 49
udp_recvspace 51
use_isno 48
Memory size problems 201
MEMORY_FACTOR 301
Mercury Interactive Corporation 397
Metadata manager location 389
migratepv 159, 164
MIN_UTIL 301
minfree 424
minfree threshold 119
minfree/maxfree 18
minperm 424, 449
minperm/maxperm 18

minpgahead 153
minpout 168
minservers 173
Mirror Write Consistency 158
Mirror Write Consistency (MWC) cache 155
Mirror write consistency check 158
Mirror write consistency record 158
Mirroring across different adapters 161
Mirroring across different adapters on different busses 161
Mirroring and striping at the same time 162
Mirroring on different disks 161
Mirroring-scheduling policies 160
mklv 164
mmdf 270
mmfsadm 270
mmlsdisk 270
mmlsfs 270
mmlsmgr 269
monitor CPU usage 212
Monitoring tools
  CPU 203
  I/O 203
  Memory 203
  Network 203
monitoring tools 23, 203
mount 71
mount command
  rsize option 71
mount process for NFS 69
mountd daemon 69
MSS 33
MTU 27, 28, 55, 195
MTU discovery settings 234
multi-level run queue 130
  Active 131
  Idle 131
  Stopped 131
  Swapped 131
  Zombie 131
Multiple client scaling problems 200
Multiple Collision Count 250, 284
Multiple paging spaces 166
multiuser workloads 7

N

Nagle Algorithm 42
  specific rules 42
Nagle Algorithmn 191
Nagle problem 200
name resolution 293
nbc_limit 60
nbc_max_cache 61
nbc_min_cache 62
nbc_pseg 62
nbc_pseg_limit 63
NBC-related tunables
  nbc_limit 60
  nbc_max_cache 61
  nbc_min_cache 62
  nbc_pseg 62
  nbc_pseg_limit 63
ndb 271, 272
netpmon 240, 275
netpmon options
  all 276
  cpu 276
  dd 276
  nfs 276
  so 276
  -t 276
  -v 277
netstat 25, 27, 28, 44, 73, 76, 99, 196, 197, 251, 272, 281
  Active connections 282
  Adapter and network statistics 284
  Communication subsystem summaries 281
  Ierrs/Oerrs 232
  Ipkts 232
  Network memory usage 285
  Opkts 232
  Protocol statistics 286
  Routing table 283
netstat -i 81
netstat -s 28
Network
  Resource tuning 410
network
  iptrace 192
network I/O statistics 241
network memory pools 30
Network monitoring tools 203
Network related tunables
  ipforwarding 54
  ipqmaxlen 57
  MTU 55
  tcp_mssdflt 54

tcp_nodelay 55
tcp_pmtu_discover 56
udp_pmtu_discover 57
network traffic considerations 106
Networking related tunables
subnetsarelocal 53
NFS 190
Cache File System 78
dropped packets 75
by the client 75
by the network 77
by the servers 76
mount process 69
timeo parameter 74
NFS client
badcalls 292
badxid 292
calls 292
cantconn 292
newcred 292
nomem 292
retrans 292
timeout 292
NFS file system 78
NFS mountpoint 291
NFS performance degradation 73
NFS server 288
badcalls 289
badlen 289
calls 289
dupchecks 290
dupreqs 290
nullrecv 289
NFS tuning 68, 201
biod daemon 69
checklist 79
examples 82
mountd daemon 69
nfsd daemon 69
Oerrs 79
read throughput 70
socket buffer overflows 80
write throughput 70
NFS Version 3 70
nfs_device_specific_bufs 65
nfs_dynamic_retrans option 74
nfs_max_threads 289
nfs_max_threads option 72
nfs_rfc1323 66
nfs_socketsize 64, 80, 82
nfs_tcp_socketsize 65, 82
nfsd 64
nfsd daemon 69
nfsd daemons 72
nfsd processes 247
nfsd threads 288
nfso 9, 45, 63
nfso command
nfs_dynamic_retrans option 74
nfs_max_threads option 72
NFS-related tunables
biod 64
nfs_device_specific_bufs 65
nfs_rfc1323 66
nfs_socketsize 64
nfs_tcp_socketsize 65
nfsd 64
nfso 63
nfsstat 287
nfsstat -cr 80
nice 8, 133, 139
nice value (NI) 219
no 9, 27, 40, 45, 46, 182, 191, 234
ARP cache tuning commands 40
arpt_killc 40
no variable
arpqsize 40
arptab_bsiz 40, 194
arptab_nb 40
ipforwarding 113
tcp_mssdflt 113
tcp_pmtu_discover 197
tcp_recvspace 90
tcp_sendspace 90
udp_pmtu_discover 197
nocache option 357
node
customization 85
gateway 200
identical workload 22
installation 85
migration 85
node groups 404
node selection
capacity 483
performance 484
nonfixed priority thread 133
nslookup 237, 293

Interactive mode 293
NSORDER 111
ntpq 295
ntptrace 295
NUMBER_OF_BALANCE 301
numperm 449

O
Oerrs 76, 79
OLAP 407
OLTP 407, 409, 450
Only Station field 250
Oracle
Cache Fusion 440
Oracle Parallel Query (OPQ) 433
Oracle Parallel Server 433
Architecture 434
Database 434
Degree 437
Distributed Lock Manager (DLM) 433, 436, 439
Instance 434, 435, 437
Oracle Parallel Query (OPQ) 433
Virtual Shared Disk (VSD) 433, 436, 438
Oracle parameters
db_block_size 450
db_buffers 451
db_writer_processes 452
db_writers 452
dbwr_io_slaves 452
disk_asynch_io 452
log_buffer 454
optimizer_mode 454
rollback_segments 454
shared_pool_size 453
sort_area_size 453
sql_trace 453
timed_statistics 454
use_async_io 452
Other GPFS considerations
387
outer edge 175, 225
outer middle 175

P
packets 23, 235
page fault 122
page in 116
page steal 116
page-replacement algorithm 122
paging 189
PAGING_SPACE_BALANCE 301
parallel 160
parallel/round robin 160
parallel/sequential 160
partially damage /etc/rc.net 84
partitioned database system 402
PDT
Files and directories 300
percent CPU time (%CPU) 219
performance 469, 472
analysis 472
analyze 470
application programming interface 470
average value 472
data stream 469
definable value 472
graphical 469
local monitor 469, 470
manage 470
monitor 469, 470
network monitor 469, 470
PTX/6000 469
remote monitor 469, 470
simple network management protocol 470
single point monitor 470
system 469
toolbox 469
tune 469
user interface 470
Performance Diagnostic Tool 295
performance monitor 472
access 472
archive 472
real time 472
single point 472
statistic 472
Performance problem checklist
AIX network tunables 190
ARP cache 190
CPU 189
I/O 190
NFS 190
Processes 189
Switch send pool 190
Traffic flow 190
Transmit queue overflows 190
Users 189

Performance problems checklist
  Paging 189
Performance Studio 397
performance tuning
  objectives 21
performance tuning cycle 5
persistent storage 117
perspectives 472
physical layer 155
ping 263, 302
Poor application modules and SQL statements 412
PowerBuilder 397
pprof 303
priority (PRI) 219
process
  resume 168
process mode switching 135
  Kernel mode 135
  User mode 135
process status (ST) 219
processes 189
processor binding (BND) 219
Protocols 235
ps 304
ps command
  other optional combinations 307
ps command options
  gvc 306
pstat 213, 214, 218, 308
PTPE 469, 470, 472
  application programming interface 472
  archive 472
  install 472
  installation 472
  monitor group 473
  node installation 473
  prerequisites 472
  scalable 472
PTX/6000 470, 472
  extension 469, 472
  install agent 471
  install server 470
  installation prerequisite 471
  server 470

Q
q_err attribute 145
Qerr bit 145
queue length value 80
Queue Overflow 284
queue_depth 145
queue_depth setting 149

R
Radview 397
RAID 0 147
RAID 0+1 162
RAID 1 147
RAID 10 148
RAID 5 148
Random write-behind 155
Rational Suite Performance Studio 397
raw logical volume 227
read throughput 70
reading/writing logical partition copies 160
real memory 208
real memory management 117
real resources 8
receive flow 24
Receive/Transmit Errors 249
recommendations for ARP cache 41
Relocating bad blocks 157
renice 8, 133, 139
renice command limitations 139
reorgvg 159
Reports generated by PDT 298
Resource monitoring
  monitoring tools 203
response time 8
retransmits 291
reverse name resolution 293
RFC 1122 38
RFC 1323 37
RFC 896 42
rfc1323 14, 44, 48, 52
rfc1323 parameter 44
rmdev 79
Rotational 143
route_expire no option 34
router
  GRF 200
routing 45
  gateway 111
  static routing 45
  switch 111
rpool 106

rpoolsize 67
RS/6000 SP 402
rsize option 71
rss 204
rss_hard 204
RTE 397
rx_que_size 16
rxbuf_pool_size 16

S

S70 attached server 481
safe asynchronous writes 70
sar 207, 215, 309
sb_max 14, 47, 80, 182
scat_gat_pages setting 151
SCHED_FIFO 131
SCHED_FIFO2 132
SCHED_FIFO3 132
SCHED_OTHER 132
SCHED_RR 132
schedtune 8, 18, 130
schedtune -t 138
Scheduler
  document initial settings 18
scheduler
  Guidelines for R and D 137
  Tuning examples using R and D 137
scheduler layer 155
scheduler queues 216
scheduling policies for logical volumes 160
scheduling policy (SCH) 219
sector 143
Secure Sockets Layer (SSL) 397
Seek 143
segments 116
  client storage 117
  persistent storage 117
  working storage 117
send flow 24
Sensitivity 184
sensitivity 259
sequential 160
Sequential write-behind 154
serial line 482
server-prompted 341
setsockopt 199
setsockopt() 39
Setting minservers and maxservers 173
shared 209
shared nothing environment 402
SIGCHLD signal 131
SIGCONT signal 131
SIGSTOP signal 131
Silly Windows Avoidance Algorithm 38
simple network management protocol 470
Single Collision Count 250, 284
Single paging space 166
single parent multiple child 197
Single server problems 200
sizing memory 208
sleep 244
slow switch 4
small packets 87
SMP 412
SNA DLC profile parameters 464
SNMP
  see simple network management protocol
SO_RCVBUF 192
SO_SNDBUF 192
SOCK_DGRAM sockets 31
SOCK_STREAM sockets 31
socket buffer overflows 25, 80, 235
Socket buffers 76
sockets 31
softlimits 204, 212
SP
  adding or changing subsystems 21
  DNS 111
  monitoring 21
  network topology 111
  tuning files 6
SP environments 6
SP Ethernet 22, 482
SP Ethernet tuning 183
SP specific network tunables
  rpoolsize 67
  spoolsize 67
SP Switch 29, 44, 48, 196
SP switch
  changing MTU size 191
SP Switch 2 161
SP Switch adapter
  communication protocols 101
SP Switch attributes 102
SP switch pool sizing 201
SP Switch Router 235
SP Switch2 4, 29, 48, 196

SP Switch2 Communications Adapter 247
SP tuning 3
adapter queue overflows 96, 97
adapter tuning recommendations 99
ARP cache 193
changing adapter settings 98
commercial and database 89
characteristic 89
Ethernet network 110
high-speed tuning 86
initial tuning parameters 6
Nagle 191
network traffic considerations 106
PSSP 3.2 100
rpool 106
rpoolsize 99
sample switch pool calculation 109
Scientific and Technical 87
characteristics 87
Server Configuration 91
characteristics 92
small packets 87
Software Development 86
SP Switch2 100, 106
specific tuning recommendations 83
spool 106
spoolsize 99
transmit adapter queue 95
transmit queue 96
workload tunables summary 93
xmt_que_size 98
spmon 313
spool 106
spoolsize 67
SSA 161
SSA RAID 147
ssaconn 147
stack 204
stack_hard 204
standalone servers 3
startsrc -s 79
static routing 45
statvsd 313
stopsrc -s 79
strict_maxperm 123, 126
Striping 161
Subnet addressing 41
subnets 22
subnetting 35
Summary table for disk I/O 252
Summary table for file I/O 253
Summary table for logical volume I/O 254
summary table for socket calls 280
svmon 120, 125, 207, 314
svmon -dlC 208
svmon -dU 211
Switch adapter 233
switch running slow 4
Switch send pool 190
syncd daemon 155
Synchronization point file server accesses 201
syncvg 159
system
overhead 470
resource 470
resource statistic 470
system partitioning 482

T

tar 168
TCP acknowledgement 42
TCP receive flow 26
TCP send flow 26
TCP send window 38
TCP sliding window 36
Congestion avoidance 39
Nagle Algorithm 39
Silly Window Syndrome 38
Slow start 39
TCP sliding window improvements 38
TCP sliding windows
Delayed ACK 39
TCP statistics 236
TCP/IP
performance 86
TCP/IP concepts 23
tcp_mssdflt 14, 34, 48, 182, 195
TCP_NODELAY 39, 42, 275
tcp_nodelay 48, 55
tcp_pmtu_discover 14, 56, 200
tcp_recvspace 14, 36, 48, 50, 182, 195
optimal settings 196
TCP_RFC1323 275
tcp_sendspace 14, 36, 48, 49, 182, 195
optimal settings 196
TCP_STDURG 275
tcpdump 238, 239, 263, 317

technical applications 481
thewall 14, 31, 46, 182, 197
thin node 481
thread 129
Thresholds 300
throughput 8
timeo parameter 74
timeouts 291
Tivoli Storage Manager 337
    Client CPU options
        COMPRESSION 348
        RESOURCEUTILIZATION 347
    Client I/O options
        RESOURCEUTILIZATION 348
    Client memory options
        COMPRESSION 346
        MEMORYEFFICIENT 347
        TXNBYTELIMIT 347
    Client network options
        TCPBUFFSIZE 346
        TCPNODELAY 346
        TCPWINDOWSIZE 346
    no parameters 339
    Scheduling options
        MAXCMDRETRIES 341
        MAXSCHEDSESSIONS 341
        QUERYSCHEDPERIOD 341
        RETRYPERIOD 341
        SCHEDMODES 341
    Server CPU options
        EXPINTERVAL 343
        MAXSESSION 343
        RESOURCEUTILIZATION 344
    Server I/O options
        MOVEBATCHSIZE 344
        MOVESIZETHRESH 344
        TXNGROUPMAX 345
    Server memory options
        BUFPOOLSIZE 342
        LOGPOOLSIZE 343
        SELFTUNEBUFPOOLSIZE 342
    Server options
        MAXSESSION 340
        TCPNODELAY 340
        TCPWINDOWSIZE 340
        USELARGEBUFFERS 341
Token Errors fields 250
Token Ring 29, 44
Token-Ring 240
Token-ring tuning 112
Token-Ring values 16
tokstat 96, 248
Tools
    netstat 281
topas 203, 217, 218, 319
topology considerations 22
Topology Services 184
tprof 320
trace facility 322
traceroute 321
track 143
traffic flow 190
Transfer 144
Transmission Control Protocol 23
transmit adapter queue
    MCA 96
    PCI 96
transmit queue 96
Transmit queue overflows 190
trcrpt 324
TREND_THRESHOLD 301
TS_Config class
    frequency 184
    sensitivity 184
tunable for the rc.net 84
Tuning
    AIX system tunable parameters 406
    AIX virtual memory parameters 406
    Batch workload 407
    Bottlenecks 407
    Check for errors 407
    CPU 408
    Disk geometry considerations 445
    Disk I/O 410
    Document RDBMS parameters 406
    DSS workload 407
    File server 21
    Hardware configurations 406
    Hot disk avoidance 445
    I/O 409
    I/O wait 409
    Latent demand 408
    Logical resource access 410
    Machine details 406
    Memory 408
    Memory - free memory 447
    Network 410
        Parameters 406

OLAP 407
OLTP 407, 409
Overworked system 408
Paging space 446
Resources 407
Sequential read ahead 446
SMP balanced CPU utilization 449
Upgrade to latest fix levels 407
Utilization 407
What can we tune? 411
Workload details 406
tuning 3
Tuning checklist for NFS 79
tuning cycle 5
Tuning DB2 UDB
Agents - maximum number of 417
Applications - db2agent usage 417
Buffer pool 414
Buffer pool hit ratio 413
Buffer pool size 413
Changed pages threshold 415
Control block information memory area 417
Database heap size 417
DMS tablespace container 419
DSS environment 413, 414
Event monitor buffers 417
I/O servers 414
Lock escalation 418
Lock list
Maximum storage 418
Locks 418
maximum number of active applications 417
Memory allocation 418
Memory disclaiming 418
OLTP environment 413, 414, 415, 417
Package cache size 416
Page cleaners 414
Parallel I/O 419
Process private memory 415
Sort heap threshold 416
Sort time 415
SQL compiler workspace 416
SQL statement caching 416
Statement heap size 416
Tablespace page size 419
Tablespace single containers 419
tuning files 6
Tuning Logical Volume Striping 163
tuning methodology 5
apply and manage new settings 9
document your system 6
establish new settings to reflect resource priorities 9
evaluate the workload 6
identify critical resources 8
installation or migration 6
minimize the workloads critical resource requirements 8
monitor performance indices 9
record and keep track of system settings 9
tuning methology
set performance objectives 7
Tuning mirror write consistency 159
Tuning Oracle
AIX buffer cache 451
AIX LVM 442
Asynchronous I/O 442
Batch workloads 453
Buffer cache
Size 447
Size for a JFS based database 448
Buffer cache size for a raw device based database 448
Disk geometry considerations 445
DSS workloads 453
Hot disk
Avoidance 445
JFS 451
JFS or raw devices 444
Logical volume creation 443
Oracle files 442
Paging
Paging space 446
Raw devices or JFS 444
Sequential read ahead 446
SGA 449
Buffer cache 451
SMP balanced CPU utilization 449
Tuning random write-behind 155
Tuning sequential write-behind 154
Tuning sequential-access read ahead 153
Tuning the Topology Services subsystem 184
tuning.cust 6
tx_que_size 16

U

UDP socket receive buffer 25

UDP socket send buffer 25
udp_pmtu_discover 14, 57
udp_recvspace 14, 25, 51
udp_sendspace 14, 25, 182
umount 165
unexpected Nagle behavior 192
unlimited 204, 212
upd_recvspace 182
Upgrades 411
use_isno 48
    tunable parameters for each available interface 48
User Datagram Protocol 23
User mode 136
user-space windows 101, 103, 104
Using the nice command 139
Using the renice command 140

V

vdidl3 108
    allocd 108
    bkt 108
    comb & freed 109
    fail 109
    free 109
    split 109
    success 109
vdidl3xx 108
    SPSMX 108
    TB3 108
    TB3PCI 108
Virtual Memory Management
    page in 116
    page steal 116
    segments 116
Virtual Memory Manager 30, 116
    document initial settings 18
Virtual Shared Disk (VSD) 433, 438
Visual Basic 397
Visual C++ 397
vmstat 73, 130, 205, 207, 213, 324
vmstat output 214
vmtune 9, 18, 120, 153, 154, 205, 206
VSD 353, 438
    node down indication 184
    SP switch considerations
        maximum buddy buffer size 356
        maximum IP message size 356
        mbufs and thewall 355
        send/receive pools 355
VSD and KLAPI 364
VSD performance parameters
    buddy buffers 359
    Maximum I/O request size 361
    VSD cache buffer 357
    VSD pbufs 358
    VSD request blocks 358
VSD tunable areas 354
VSD tuning recommendations 362

W

Web server 394
WebLoad 397
WebSphere Application Server Engine 394
WebSphere Application Server Resource Analyzer 398
WebStone 396
wide node 481
win_maxsize 101, 102
win_minsize 101, 102
win_poolsize 101, 102
window 36, 101
    IP 101
    Service 101
    User Space 101
    VSD 101
window sizes 100
    win_minsize 101
windows sizes
    win_maxsize 101
    win_poolsize 101
working storage 117
Workload Manager 8
write throughput 70
Write Verify 161
write_queue_mod setting 150
write-behind 154
    random 154
    sequential 154

X

xmperf 210, 230, 241
xmt_que_size 98
    recommended setting 99

Z

zero copy transport protocol 365
zombie state 131

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at **ibm.com**/redbooks
- Fax this form to: USA International Access Code + 1 845 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number **Redbook Title**	SG24-5340-01 RS/6000 SP System Performance Tuning Update
Review	
What other subjects would you like to see IBM Redbooks address?	
Please rate your overall satisfaction:	O Very Good O Good O Average O Poor
Please identify yourself as belonging to one of the following groups:	O Customer O Business Partner O Solution Developer O IBM, Lotus or Tivoli Employee O None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	O Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. **ibm.com**/privacy/yourprivacy/

IBM

Redbooks

RS/6000 SP System Performance Tuning Update

RS/6000 SP System Performance Tuning Update

What you always wanted to know about SP performance but were afraid to ask

New tuning information about PSSP 3.2 and AIX 4.3.3

Sample tuning scripts included

This IBM redbook provides an updated version of the SP System Performance Tuning Guide. It now includes new information about changes in AIX V4.3.3 and PSSP V3.2 that can affect your SP System performance. Also, new hardware that is available especially for the SP, such as the new SP Switch2, is described along with its enhancements and tunables.

This redbook now includes information about certain applications: Tivoli Storage Manager (ADSM), DB2, and GPFS, just to name a few. To make this book comprehensive, we added more specific information about monitoring your system to increase performance. This also includes some useful scripts that can help you improve system performance or analyze the current settings of your system faster and easier.

The reader of this book can range from a novice of performance tuning to an experienced administrator. Considerable thought has gone into the layout of the book. Readers with greater experience in SP tuning have new "quick reference" features, such as startmaps and part guides: this will make referencing parameter and other environment information easier. Full and clear explanation of concepts and procedures are still provided.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-5340-01 ISBN 0738419281