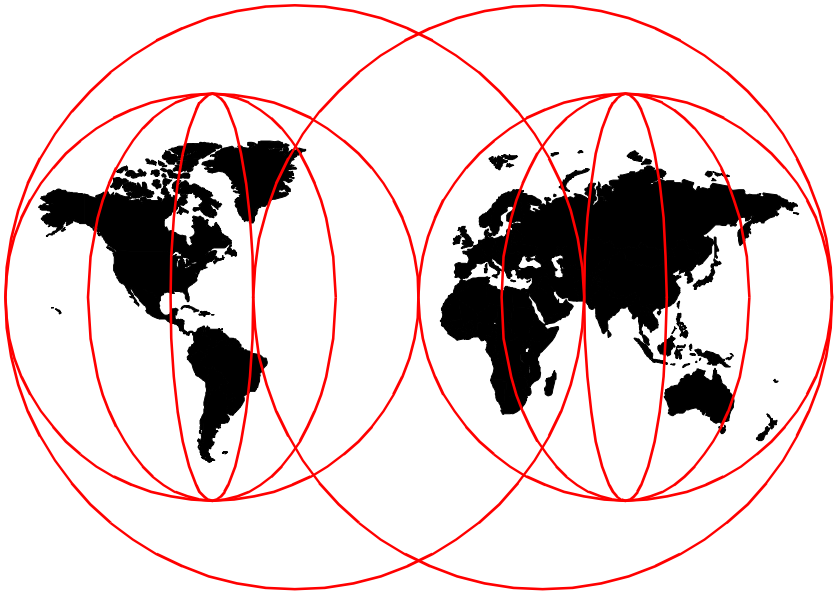
Sizing and Tuning GPFS

Marcelo Barrios, Terry Jones, Scott Kinnane, Mathis Landzettel
Safran Al-Safran, Jerry Stevens, Christopher Stone, Chris Thomas, Ulf Troppens



International Technical Support Organization

www.redbooks.ibm.com

SG24-5610-00

SG24-5610-00

International Technical Support Organization

Sizing and Tuning GPFS

September 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix B, “Special notices” on page 277.

First Edition (September 1999)

This edition applies to GPFS Version 1, Release 2 (5765-B95) for use with the AIX Operating System Version 4, Release 3 Modification 2 and to Version 3 Release 1 of ADSTAR Distributed Storage Manager for AIX (5765-C43).

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Mail Station P099
522 South Road
Poughkeepsie, NY 12601-5400

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures ix

Tables xiii

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

Chapter 1. GPFS architecture 1
1.1 General concepts 1
1.1.1 Architecture overview 2
1.2 Data flow and potential bottelnecks 4
1.2.1 Write data flow 4
1.2.2 Read data flow 12
1.2.3 Potential bottlenecks on writes and reads 15
1.3 GPFS software structure and required services 17
1.3.1 Internal data structures 17
1.3.2 Configuration Manager 19
1.3.3 Stripe Group Manager 19
1.3.4 Metanode 20
1.3.5 Token Manager Server 20
1.3.6 VSD 20
1.3.7 Recoverable Virtual Shared Disks 23
1.3.8 SP Switch 25
1.3.9 Clustering subsystems 35
1.3.10 System Data Repository 37
1.4 GPFS operation 38
1.4.1 User interfaces 38
1.4.2 Security 39
1.4.3 Consistency 39
1.4.4 Failure and recovery 39
1.5 Positioning GPFS and other file systems 45
1.5.1 Comparison of GPFS with other file systems 45
1.5.2 GPFS advantages 51
1.5.3 GPFS limitations 54

Chapter 2. Application considerations 57
2.1 GPFS application block Size 57
2.2 GPFS application performance 58
2.3 GPFS application I/O 58
2.3.1 Sequential and random application I/O 58

2.3.2 Serial and parallel file system I/O 59
2.3.3 Application I/O patterns and GPFS 60
2.3.4 Exploiting GPFS read prefetch and write behind 61
2.4 Data partitioning 61
2.4.1 Round-robin or segmented? 61
2.4.2 One file or multiple files? 62
2.4.3 Using files larger than two gigabytes 63
2.5 MPI-IO and GPFS 64
2.5.1 About MPI-IO 64
2.5.2 Local to global transformations 66
2.5.3 Application buffering 66
2.5.4 Hints support 66
2.6 On designing other I/O libraries with GPFS 66
2.6.1 Portability concerns 67
2.6.2 Exposing GPFS internals 67
2.6.3 Threads, signals, and communication issues 68
2.7 Analyzing an applications I/O 69
2.7.1 AIX trace 69
2.7.2 Pablo 69
2.7.3 Monitoring file system activity 69

Chapter 3. Sizing GPFS 71
3.1 Sizing concepts 71
3.1.1 Data 72
3.1.2 Metadata 76
3.1.3 Servers 80
3.2 Sizing methodology 85
3.2.1 Step 1 - Find file system requirements 86
3.2.2 Step 2 - Recoverability considerations 88
3.2.3 Step 3 - Determine minimum number and type of disks 89
3.2.4 Step 4 - Check and adjust for file system capacity 90
3.2.5 Step 5 - Determine number of VSD servers required 90
3.2.6 Step 6 - Determine VSD server configuration 93
3.2.7 Sizing example 1 94
3.2.8 Sizing example 2 97

Chapter 4. Tuning GPFS 101
4.1 Isolating and identifying problems and bottlenecks 102
4.1.1 Working from the dataflow diagram 103
4.1.2 High impact issues 103
4.1.3 Medium impact issues 120
4.1.4 Low impact issues 128
4.2 Tuning verification 134

4.2.1 Monitoring at the server 135
4.2.2 Monitoring at the client 144
4.2.3 Interpreting the numbers. 148
4.3 Tuning case studies 150
4.3.1 Hardware, software, and GPFS configuration 151

Chapter 5. Implementing and tuning ADSM for GPFS 165
5.1 ADSM relevant differences between JFS and GPFS 165
5.1.1 Functional differences 166
5.1.2 Impact of the delayed update of mtime 167
5.1.3 Performance differences. 168
5.1.4 Data volumes 168
5.2 Resource requirements 169
5.2.1 Resource requirements for VSD 170
5.2.2 Resource requirements for GPFS. 170
5.2.3 Resource requirements for ADSM 171
5.3 Case studies 171
5.3.1 Test system configuration. 172
5.3.2 Test methodology 173
5.3.3 ADSM and TCP/IP configuration. 174
5.3.4 Configuration 1: ADSM client on single VSD client node. 175
5.3.5 Configuration 2: ADSM client and server on same SP node . . . 180
5.3.6 Configuration 3: Using multiple ADSM client nodes 183
5.3.7 Configuration 4: ADSM clients on VSD server nodes 187
5.3.8 Impact of tuning maxFilesToCache. 189
5.3.9 Comparison of ADSM performance between JFS and GPFS . . 191
5.3.10 Full Incremental versus selective backup 192
5.3.11 Restore versus replace. 193
5.4 Recommendations 195
5.4.1 Which SP Node to use as an ADSM server 195
5.4.2 Which SP node to use as an ADSM client 196
5.4.3 Is there any advantage on running multiple client sessions?. . . 197
5.4.4 How many SP Nodes be used as an ADSM clients?. 197

Chapter 6. Test results. 199
6.1 Base run tests 199
6.1.1 Serial tests 200
6.1.2 Parallel tests. 200
6.1.3 Random tests 200
6.1.4 Configurations 200
6.1.5 Applications 204
6.1.6 Measurement tools 205
6.1.7 Measurements 206

6.2 RAID-5 array size tests 206
6.2.1 Measurements 206
6.3 Metadata tests 207
6.3.1 Measurements 207
6.4 Client max throughput tests 208
6.4.1 Measurements 208
6.5 Analysis 208
6.5.1 Compare RAID-5, mirroring, replication, and JBOD 208
6.5.2 Compare SSA disk and loop combinations 210
6.5.3 Compare serial parallel and random application performance. . 212
6.5.4 Compare RAID-5 with different array sizes 214
6.5.5 Compare RAID-5 with/without metadata on RAID-5 215
6.5.6 Investigation of maximum client data throughput 215
6.5.7 Analysis of CPU usage with regard to dedicated VSD servers . 216
6.5.8 How the number of VSD servers affects performance. 218
6.5.9 Validating sizing 220
6.6 Conclusions 221
6.6.1 File system options. 221
6.6.2 SSA disk subsystems 221
6.6.3 Sequential I/O versus random 222
6.6.4 Metadata and RAID-5 222
6.6.5 Dedicated VSD servers 222
6.6.6 General conclusions 222

Appendix A. Measurements 225
A.1 Base runs 225
A.1.1 Serial. 225
A.1.2 Parallel 246
A.1.3 Random 266
A.2 RAID-5 array size tests 271
A.2.1 7+P RAID-5 S2_C1_Tests3 271
A.2.2 15+P RAID-5 S2_C1_Tests3 272
A.3 Metadata tests 273
A.3.1 4+P RAID-5 combined data and metadata S2_C1_Tests3 273
A.4 Client max throughput tests 275

Appendix B. Special notices 277

Appendix C. Related publications 281
C.1 International Technical Support Organization publications. 281
C.2 Redbooks on CD-ROMs 281
C.3 Other publications 281

How to get ITSO redbooks 283
IBM Redbook fax order form 284

List of abbreviations 285

Glossary 289

Index 295

ITSO Redbook evaluation 301

Figures

1. GPFS architecture . . . 3
2. Control and data flow for a write operation . . . 5
3. Control and data flow for a read operation. . . . 13
4. Potential bottlenecks to GPFS data flow.. . . . 16
5. VSD states and associated commands . . . 22
6. VSD architecture . . . 23
7. Recoverable VSD . . . 24
8. VSD take over . . . 25
9. Switch initialization process . . . 26
10. IP to switch interaction . . . 33
11. Kernel structures for IP over switch communications . . . 34
12. State transistions for RVSD recovery . . . 43
13. JFS offers local access to data only . . . 46
14. The complexity of NFS mount points . . . 48
15. The DFS server bottleneck . . . 50
16. The scalability of GPFS . . . 52
17. Comparing sequential and random I/O . . . 59
18. Comparing serial and parallel I/O . . . 59
19. Parallel file system I/O . . . 60
20. Segmented and round-robin file layouts . . . 62
21. MPI-IO file types . . . 65
22. How an entire MPI-IO view is formed with file types . . . 65
23. Diagram showing I/O library layers . . . 67
24. Adapter communication windows . . . 68
25. Performance of GPFS reads with block size . . . 87
26. Performance of GPFS writes with block size . . . 88
27. Example 1 SSA loop disk layout for a pair of connected nodes . . . 96
28. Example 2 SSA loop disk layout for a pair of connected nodes . . . 100
29. Worker thread performance example . . . 148
30. Flow of symptoms within a GPFS architecture . . . 149
31. Hardware configuration for GPFS . . . 152
32. Bandwidth versus block size . . . 164
33. Example implementation for ADSM on GPFS . . . 170
34. Cabling and volumes of ADSM Server . . . 172
35. Cabling of VSD servers . . . 173
36. Multiple ADSM Client Sessions on single VSD Client Node . . . 176
37. Backup data throughput . . . 178
38. Reference Configuration: ADSM Server CPU Utilization - Backup . . . 178
39. Restore data throughput . . . 179
40. ADSM server CPU utilization - Restore . . . 179

41. Configuration 2: Multiple ADSM client sessions on ADSM server node . 180
42. Rel. backup data throughput: 256 MB files (IP and shared memory) . . . 182
43. Rel. backup data throughput: 10 KB files (IP and shared memory). 182
44. ADSM server CPU utilization — Backup (shared memory to IP). 183
45. Configuration 3: Multiple ADSM/VSD client sessions and nodes 184
46. Rel. backup data throughput: ADSM client running on four SP Nodes . . 185
47. Rel. restore data throughput: ADSM client running on four SP Nodes . . 185
48. Rel. backup data throughput: ADSM client running on two SP Nodes . . 186
49. Rel. restore data throughput: ADSM client running on two SP Nodes. . . 186
50. Conf. 4: Multiple ADSM client sessions on multiple VSD server nodes. . 187
51. Rel. backup data throughput: ADSM client on one VSD server. 188
52. Rel. backup data throughput: ADSM client on two VSD servers 188
53. Backup data throughput: Ratio of maxFilesToCache 10,000 to 200 190
54. Restore data throughput: Ratio of maxFilesToCache 10,000 to 200. . . . 190
55. Backup data throughput: Ratio of JFS to GPFS 191
56. Backup data throughput: JFS/GPFS — Increased maxFilesToCache . . 192
57. Backup data throughput: Ratio of selective to full Incremental 193
58. Restore data throughput: Ratio of replace to restore: Single SP node . . 194
59. Restore data throughput: Ratio Replace/Restore: Multiple SP nodes. . . 195
60. SSA disk layout 1 — One SSA drawer per node. 203
61. SSA disk layout 2 — Two SSA drawers per node. 204
62. Client comparison: GPFS file system types — 4 VSD / 4 GPFS nodes . 209
63. Server comparison: GPFS file system types — 4 VSD / 4 GPFS nodes. 210
64. Server performance with 4 GPFS clients and 2 Servers JBOD. 211
65. Server performance: 1 loop per adapter — 4 clients / 2 Servers JBOD . 212
66. Performance graph s1.c4.t3 — Client View: Sequential 213
67. Performance graph s1.c4.t3 — Client View: Parallel. 213
68. Performance graph s1.c4.t3 — Client View: Random 214
69. RAID-5 server view performance with different arrays sizes — s2.c1.t3. 215
70. Results for two-client performance of S1_C4_T1 and S1_C4_T2. 219
71. Server-constrained and client-constrained imbalances 220
72. Performance graph s1.c4.t1 — Client view 226
73. Performance graph s1.c4.t1 — Server view 226
74. Performance graph s1.c4.t2 — Client view 227
75. Performance graph s1.c4.t2 — Server view 227
76. Performance graph s1.c4.t3 — Client view 228
77. Performance graph s1.c4.t3 — Server view 228
78. Performance graph s1.c4.t4 — Client view 229
79. Performance graph s1.c4.t4 — Server view 229
80. Performance graph s1.c4.t5 — Client view 230
81. Performance graph s1.c4.t5 — Server view 230
82. Performance graph s1.c2.t1 — Client view 231
83. Performance graph s1.c2.t1 — Server view 231

84. Performance graph s1.c2.t2 — Client view 232
85. Performance graph s1.c2.t2 — Server view 232
86. Performance graph s1.c2.t3 — Client view 233
87. Performance graph s1.c2.t3 — Server view 233
88. Performance graph s1.c2.t4 — Client view 234
89. Performance graph s1.c2.t4 — Server view 234
90. Performance graph s1.c2.t5 — Client view 235
91. Performance graph s1.c2.t5 — Server view 235
92. Performance graph s1.c3.t1 — Client view 236
93. Performance graph s1.c3.t1 — Server view 236
94. Performance graph s1.c3.t2 — Client view 237
95. Performance graph s1.c3.t2 — Server view 237
96. Performance graph s1.c3.t3 — Client view 238
97. Performance graph s1.c3.t3 — Server view 238
98. Performance graph s1.c3.t4 — Client view 239
99. Performance graph s1.c3.t4 — Server view 239
100.Performance graph s1.c3.t5 — Client view 240
101.Performance graph s1.c3.t5 — Server view 240
102.Performance graph s2.c1.t1 — Client view 241
103.Performance graph s2.c1.t1 — Server view 241
104.Performance graph s2.c1.t2 — Client view 242
105.Performance graph s2.c1.t2 — Server view 242
106.Performance graph s2.c1.t3 — Client view 243
107.Performance graph s2.c1.t3 — Server view 243
108.Performance graph s2.c1.t4 — Client view 244
109.Performance graph s2.c1.t4 — Server view 244
110.Performance graph s2.c1.t5 — Client view 245
111.Performance graph s2.c1.t5 — Server view 245
112.Performance graph s1.c4.t1 — Client view 246
113.Performance graph s1.c4.t1 — Server view 246
114.Performance graph s1.c4.t2 — Client view 247
115.Performance graph s1.c4.t2 — Server view 247
116.Performance graph s1.c4.t3 — Client view 248
117.Performance graph s1.c4.t3 — Server view 248
118.Performance graph s1.c4.t4 — Client view 249
119.Performance graph s1.c4.t4 — Server view 249
120.Performance graph s1.c4.t5 — Client view 250
121.Performance graph s1.c4.t5 — Server view 250
122.Performance graph s1.c2.t1 — Client view 251
123.Performance graph s1.c2.t1 — Server view 251
124.Performance graph s1.c2.t2 — Client view 252
125.Performance graph s1.c2.t2 — Server view 252
126.Performance graph s1.c2.t3 — Client view 253

127.Performance graph s1.c2.t3 — Server view 253
128.Performance graph s1.c2.t4 — Client view 254
129.Performance graph s1.c2.t4 — Server view 254
130.Performance graph s1.c2.t5 — Client view 255
131.Performance graph s1.c2.t5 — Server view 255
132.Performance graph s1.c3.t1 — Client view 256
133.Performance graph s1.c3.t1 — Server view 256
134.Performance graph s1.c3.t2 — Client view 257
135.Performance graph s1.c3.t2 — Server view 257
136.Performance graph s1.c3.t3 — Client view 258
137.Performance graph s1.c3.t3 — Server view 258
138.Performance graph s1.c3.t4 — Client view 259
139.Performance graph s1.c3.t4 — Server view 259
140.Performance graph s1.c3.t5 — Client view 260
141.Performance graph s1.c3.t5 — Server view 260
142.Performance graph s2.c1.t1 — Client view 261
143.Performance graph s2.c1.t1 — Server view 261
144.Performance graph s2.c1.t2 — Client view 262
145.Performance graph s2.c1.t2 — Server view 262
146.Performance graph s2.c1.t3 — Client view 263
147.Performance graph s2.c1.t3 — Server view 263
148.Performance graph s2.c1.t4 — Client view 264
149.Performance graph s2.c1.t4 — Server view 264
150.Performance graph s2.c1.t5 — Client view 265
151.Performance graph s2.c1.t5 — Server view 265
152.Performance graph s1.c4.t1 — Client view 266
153.Performance graph s1.c4.t1 — Server view 266
154.Performance graph s1.c4.t2 — Client view 267
155.Performance graph s1.c4.t2 — Server view 267
156.Performance graph s1.c4.t3 — Client view 268
157.Performance graph s1.c4.t3 — Server view 268
158.Performance graph s1.c4.t4 — Client view 269
159.Performance graph s1.c4.t4 — Server view 269
160.Performance graph s1.c4.t5 — Client view 270
161.Performance graph s1.c4.t5 — Server view 270
162.RAID-5 15+P performance graph s2.c1.t2 — Server view 272
163.Combined metadata and data RAID-5 4+P s2.c1.t3 — 4 Clients view . . 273
164.Separated metadata and data RAID-5 4+P s2.c1.t3 — 4 Clients view . . 273
165.Separated metadata and data RAID-5 4+P s2.c1.t3 — 1 Client view . . . 274
166.Combined metadata and data JBOD s2.c1.t2 — 1 Client view 274
167.Single node multi application s2.c1.t2 —1 Node client view 275

Tables

1. Internal and external components to GPFS 4
2. Switch adapter type and corresponding firmware information. 27
3. Comparison of file system features 51
4. Disk performance. 80
5. Performance of SSA adapters 81
6. SP Switch and GPFS throughput. 82
7. SSA disk identification 90
8. GPFS throughput performance of SSA adapters. 91
9. Table of default sizing parameters to use for GPFS 93
10. Table of size dependent initial parameters to use for GPFS 94
11. Table showing send bucket size to buffer size 137
12. Tuning verification parameters for the Switch adapter. 139
13. Tuning verification parameters for the IBM Virtual Shared Disk server . . 142
14. Tuning verification parameters for the disk sub-system. 144
15. Worker thread performance example 146
16. Initial parameter of the system. 152
17. Tuning Results 163
18. ADSM relevant functional differences between JFS and GPFS 166
19. Resource requirements for GPFS and ADSM components. 169
20. Sizing of test data 173
21. ADSM server options 174
22. ADSM client options. 174
23. TCP/IP parameters for all SP Nodes 175
24. GPFS file system parameters 175
25. Backup and restore throughput for reference configuration. 176
26. Relative data throughput using TCP/IP and shared memory. 181
27. Effect of maxFilesToCache on full incremental backups 189
28. Test environment — Setups 202
29. Test environment — Configurations 202
30. Test environment — Tests. 202

Preface

This redbook provides in-depth information about General Parallel File System (GPFS) Version 1.2. It is written for RS/6000 professionals looking for help in planning, installing, configuring, and tuning GPFS-based solutions.

The book discusses architecture and application considerations that will help you understand the performance impact that GPFS components may have on applications and how applications may avoid such pitfalls and take advantage of GPFS's parallel features. Included is a step-by-step guide for sizing a GPFS configuration and a tuning section that lists and explains parameters and software components sorted by their impact on performance and availability.

This redbook also provides recommendations and considerations for back up and recovery of GPFS using Adstar Distributed Storage Management (ADSM).

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.

Marcelo Barrios is a project leader at the International Technical Support Organization, Poughkeepsie Center. He has been with IBM since 1993 working in different areas related to RS/6000. Currently, he focuses on RS/6000 SP technology by writing redbooks and teaching IBM classes worldwide.

Terry Jones is a computer scientist at Lawrence Livermore National Laboratory in Livermore, California. He has over 10 years of experience in the high performance computing field and is currently working in the area of parallel file systems. He holds a M.S. degree in Computer Science from Stanford University and a B.S. degree in Physics from SWOSU.

Scott Kinnane is a system engineer at Petroleum Geo Services (PGS) in Perth, Australia. He has over five years experience with numerous UNIX operating systems and architectures including systems from Sun, IBM, SGI and DEC, and Linux. He has experience in designing, implementing, and tuning TCP/IP, ATM LANE, Ethernet, and ATM networks, and over twelve years of programming with numerous languages. He holds a B.S. degree in Computer Science from Curtin University, Perth, Australia.

Mathis Landzettel is a project leader at the International Technical Support Organization, San Jose Center. He joined IBM in 1994 after completing his degree in mathematics at the Technical University of Darmstadt. He writes extensively and teaches IBM classes worldwide on all areas of ADSM. Before joining the ITSO in 1998, Mathis worked in the ADSM development department in Mainz, Germany, as a software test team leader.

Safran Al-Safran is a system analyst at Saudi ARAMCO in Saudi Arabia where he has worked for eight years. He has over seven years of experience in AIX, IRIX, and Solaris field. His areas of expertise include SP planning, installation, and support. He holds B.S and M.S degrees in Computer Science from King Fahad University of Petrolume and Minerals, Saudi Arabia.

Jerry Stevens has a BSc degree in Mathematics from Exeter University and over 20 years of IT experience. He is currently an IT specialist working in an RS/6000 and SP consultancy practice in IBM, UK. Before joining IBM in 1997, Jerry worked for Shell for eight years as a systems engineer performing a range of technical consultancy and development roles and working with a variety of Open Systems platforms and architectures.

Chris Stone works for IBM in the UK providing customer services in the field of high performance computing and storage solutions. He has three years experience working with AIX and the SP. He has been with IBM since 1981 and has a wide range of experience including both analogue and digital circuit design for displays and weapons systems, software design and development in retail and communications fields, and also education delivery and systems support in many different countries. He is a chartered engineer, a member of the IEE, and holds a BSc degree in Electrical and Electronic Engineering from Bristol University.

Christopher Thomas is a Senior IT Specialist in the UK. He joined IBM in 1984 after completing his degree in Electronic Engineering at University College, London. He has been working with ADSM and other IBM storage products for the last five years and has experience with large-scale data backup and archive systems. Before that, he worked in the storage development group at IBMs Hursley Laboratory developing real-time software for use in the development and manufacture of disk drive products.

Ulf Troppens is working in ADSM development and test for IBM in Mainz, Germany. He has over ten years of experience with different UNIX systems and distributed systems. His areas of expertise include DCE, Computer Supported Cooperative Work, and real-time multimedia applications. He holds a M.S. degree in Computer Science from the University of Karlsruhe, Germany.

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

IBM Poughkeepsie
Lyle Gayne
Bob Curran
Kevin Gildea
Frank Mangione
Brian Herr
Surya Panigrahy

IBM San Jose
Nancy Young

IBM Almaden
Dan McNabb
Roger Askin

IBM Germany
Christian Bolik

We would also like to express our gratitude toward the people of the benchmark center in Poughkeepsie, in special to Bob Davis, who helped us on getting the proper equipment to run most of the performance tests.

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 “ITSO Redbook Evaluation” on page 323 to the fax number shown on the form.
- Use the online evaluation form found at `http://www.redbooks.ibm.com/`
- Send your comments in an internet note to `redbook@us.ibm.com`

Chapter 1. GPFS architecture

The General Parallel File System (GPFS) is a scalable file system developed at IBM for general-purpose usage on RS/6000 SP servers. In this chapter, we describe the architecture of GPFS and how it works. We conclude with a comparison of GPFS to other file systems.

1.1 General concepts

GPFS is the successor to the PIOFS file system. It is a high performance file system designed for the needs of sophisticated parallel programs.

Perhaps the most prominent feature of GPFS is the degree of its scalability. By *scalable*, we mean that incremental improvements, such as increased writing and reading performance, may be made to the file system by adding additional hardware of the same, or even lesser, capability. A truly scalable file system should provide high capacity and high throughput. GPFS file systems may be multiple Terabytes in capacity and provide throughput of multiple GB/sec to and from one file. As an AIX file system, GPFS inherently supports large files that exceed two gigabytes in size.

It is *general-purpose* in that it is suitable for many kinds of workloads including commercial and technical tasks.

It provides *global access* (or uniform access) to files. That is, it is possible to mount a GPFS file system from every node on an SP system making applications much easier to write.

Most users prefer solutions that offer *portability*. That is, a solution that is widely accepted and implemented by multiple vendors. Solutions that are limited to a specific hardware type or a specific software environment are generally considered much less desirable than portable solutions. The application programming interface (API) to GPFS is a portable solution based on the POSIX standard; no GPFS-specific system calls are needed or provided. This means that applications using standard POSIX functions will run on GPFS without modification or re-compilation[1]. Furthermore, AIX utilities work with GPFS.

One final consequence of a POSIX interface is *ease of use*. Since POSIX is well known, initial use is easier for most developers.

[1] Several common non-POSIX functions are not provided with GPFS 1.2: mmap, munmap, and msync. In addition, since the atime/mtime/ctime information is maintained in a distributed manner (for performance reasons), some time is required before the most up-to-date information on an actively changing file is available to all nodes.

GPFS incorporates several techniques to enhance *reliability*. GPFS has extensive recovery mechanisms. For example, the file system keeps track of which nodes are available and dynamically adjusts to migrate file system tasks away from non-responding nodes.

In the parallel programming environments common to SP systems, multiple tasks of a parallel job frequently need to write or read from a single file from multiple nodes. Unfortunately, the UNIX file system does not support this activity well: The mechanisms that it provides for file consistency (file locking) are performed at the entire file level. GPFS addresses this key issue by providing *byte-range locking*. That is, one task may be granted write or read access to a portion of a file, and other tasks may be granted write or read access to other portions of the same file. This permits writes and reads to occur concurrently without serialization because of consistency.

The importance of effective caching to file system performance cannot be overstated. GPFS provides extensive *read-ahead* (prefetch) and *write-behind* caching. When reading, GPFS analyzes the read access pattern and attempts to bring in data that is speculated to be needed in the future prior to the actual read call. Similarly, small sequential writes are buffered in a write-behind cache until an efficient writing size is collected. Caching is performed at the application nodes. The analysis is performed on a per file handle basis. If one process opens the same file twice, GPFS can use different caching strategies for each instance.

1.1.1 Architecture overview

GPFS is implemented as a number of separate software subsystems, each of which may be distributed across multiple nodes within an SP system. Figure 1 on page 3 shows two nodes from a typical GPFS configuration as dark-filled boxes. The right node is an *application node.* That is, a node which has mounted a GPFS file system and running a user application that accesses that file system. The left node is a *VSD server node.* A VSD server node is one that physically has a number of disk drives attached that may be shared with other nodes in the same partition.

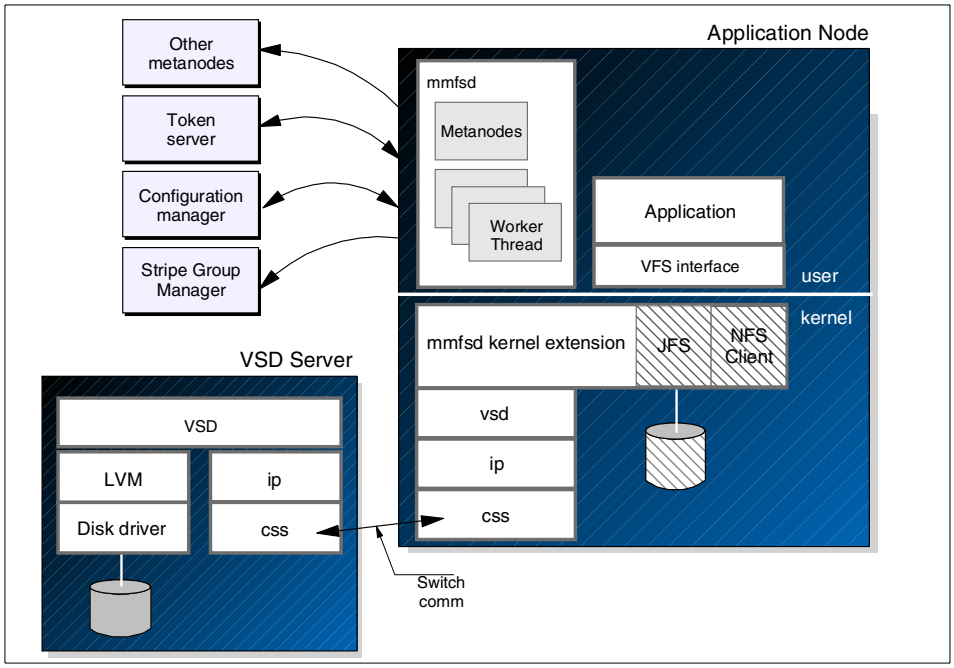


Figure 1. GPFS architecture

In contrast to the dark-filled boxes in Figure 1, which represent different nodes, the light-filled boxes represent the different software subsystems and services utilized in GPFS. The figure shows that several software subsystems utilized by GPFS must be present on the application node(s); other software subsystems may be on a node other than the application.

These subsystems are described in more detail in 1.3, “GPFS software structure and required services” on page 17. At this point, we merely call your attention to the fact that some of the subsystems are internal to GPFS, while other subsystems are provided as external services utilized by GPFS. Internal services are provided by the GPFS daemon, mmfsd, while external services are provided by other daemons and subsystems. For example, GPFS utilizes the SP Switch subsystem to perform communications. The SP Switch subsystem is a general subsystem used for many services; it is not

part of GPFS. Table 1 lists those services that are internal to GPFS as well as the external subsystems and services utilized by GPFS.

Table 1. Internal and external components to GPFS

Software subsystems and services **internal** to GPFS daemon (mmfsd)	Software subsystems and services **external** to GPFS
Configuration Manager	Virtual Shared Disk (VSD)
Stripe Group Manager	Recoverable Virtual Shared Disk (RVSD)
Metanode	SP switch subsystem
Token Manager Server	Group Services
	System Data Repository (SDR)

1.2 Data flow and potential bottelnecks

One informative way to study file systems is to do an analysis of data flow for reads and writes. This is particularly true of file systems with distributed components. What follows is such an analysis plus some information on potential bottlenecks.

1.2.1 Write data flow

Figure 2 on page 5 shows how GPFS interacts with other system components during write operations. We will consider a write of 256 KB that we assume is the size of one full GPFS block. If the write is smaller than a GPFS block, GPFS may utilize a write-behind strategy for better performance.

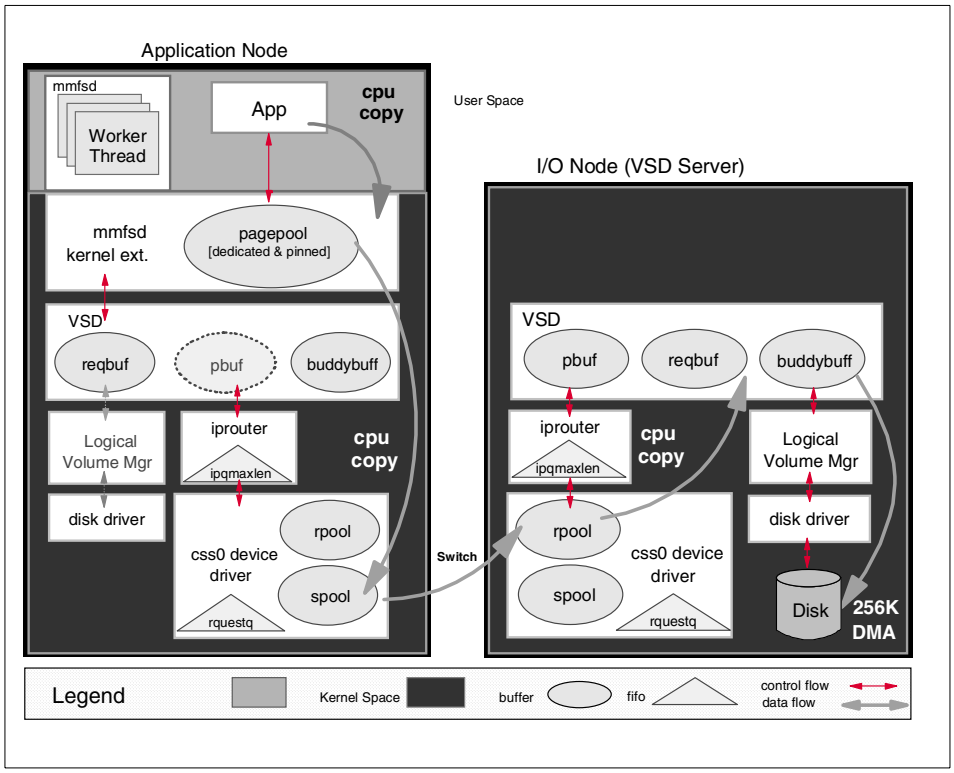


Figure 2. Control and data flow for a write operation

The following steps explain the write flow on the system:

1. The application makes a write call with a pointer to a buffer in its space.
2. The mmfsd on the application node checks to see if it holds an Exclusive lock for the file. That is, it checks to see if it has the right to modify the file. If this is the first write for this node and for this file, a write token must be acquired; otherwise, (if an Exclusive token is already held) skip to step 5.

   If the file already open is local, the GPFS daemon (mmfsd) checks if this file is already granted to another application, either local or remote. If not, mmfsd gets a lock and then informs the Token Manager Server about the request. If the file has already been granted to other nodes, mmfsd will negotiate with those nodes in order to get the requested token on the first read or write.

   If the file already open is remote, mmfsd contacts the Token Manager Server for the file system to request a token. If other nodes already have the token for this file, the Token Manager Server will send back the list of

nodes having the token. It is the responsibility of the requesting mmfsd to negotiate with the nodes in that list to obtain a token. This technique is employed for scalability reasons: Distributing this task to the mmfsd reduces serialization at the Token Manager Server.

The request is for an Exclusive access token. The Exclusive request contains both a *required-byte- range* and a *desired-byte-range* for the file. The optimistic algorithm used by GPFS usually requests all subsequent bytes in the file for the desired-byte-range.

3. The Token Manager Server determines if any conflicts exist for the required-byte-range Exclusive request. Assuming there is no Exclusive access token currently held for the required-byte-range, the Token Manager Server will grant an Exclusive access token for the required-byte- range and as much as possible of the desired-byte-range. If future tasks require write access to a different non-overlapping section of the file, an exclusive token with a byte range qualifier is always granted if there is not a conflict (see 1.4.3, “Consistency” on page 39).
4. The mmfsd receives the Token Manager Server response containing the Exclusive token and a specified byte range.
5. The mmfsd on the application node acquires some of the metadata that it will need to update when the file increases in size. Specifically, it gets some unused disk-address-pointers in either the file’s inode, or if all disk-address-pointers have been used in the inode, some unused disk-address-pointers in the file’s next indirect block. These metadata structures are then sent to the file’s metanode if this is remote.
6. GPFS acquires a location on disk for the data to be written. At mount time, each GPFS node is given an *allocation segment,* which is a cluster of available disk blocks. Each allocation segment contains blocks from every disk. To reduce hot spots, the blocks are ordered such that they address disks in a round-robin order. When all of the disk blocks in an allocation segment are in use, the mmfsd daemon requests another allocation segment from the Stripe Group Manager. Until the file system is 95 percent in use, the allocation service of the Stripe Group Manager employs an exhaustive search algorithm to find available space. Since this algorithm becomes too expensive as the file system becomes full, a different, less exhaustive but quicker algorithm is employed at 95 percent full. Once the file system reaches 99 percent full, a third algorithm is employed. The third algorithm is not exhaustive; if it does not find a block quickly, it returns ENOSPC.

   The next unused disk-address-pointer in the inode (or indirect block) is updated with the disk location from the allocation segment. The metanode will be informed of the allocation assignments for the

disk-address-pointers at sync time (every few minutes). All updates are done in a safe order: After the data is committed to disk, the new indirect blocks with pointers to the data may be committed to disk, then the inodes with pointers to the new indirect blocks may be committed to disk.

7. GPFS acquires a buffer from the pagepool. If there is no buffer available, a buffer is made available by writing the oldest dirty buffer out to disk.
8. The data is moved from the application's data buffer to the GPFS pagepool buffer.

pagepool

The *pagepool* is used to cache user data and indirect blocks. The default value is 20MB. It is the GPFS pagepool mechanism that allows GPFS to implement read and write requests asynchronously via read ahead and write behind mechanisms. Increasing pagepool will increase the amount of cached data available to applications. This will provide performance benefits in applications that do large amounts of I/O quickly and in applications where re-use of data is high. Its setting can also be particularly critical for applications that do random I/O.

9. GPFS schedules a worker thread to continue the write if this is a full block write or somebody else needs this block. At this point, the data is in the GPFS buffer, and the application has completed the write system call.

worker1Threads

The *worker1Threads* parameter controls the maximum number of threads used for controlling sequential write behind. The minimum value for worker1Threads is 1 and the maximum 72. The default is 48.

The worker2Threads parameter controls the maximum number of threads used for controlling other operations, primarily those involving directories. The minimum value for worker2Threads is 1 and the maximum is 12. The default is the maximum, 12.

10. The GPFS thread makes a call to the VSD strategy routine requesting the data be written to disk in chunks the size of the GPFS blocksize. If the disk configuration is RAID, and the RAID stripe width is greater than the GPFS block size, a *read-modify-write* cycle at the RAID controller will be required later when the data is actually committed to rotating storage. That is, in order to write the data and the new RAID parity to disk, the unmodified

data within the stripe must be read first to recompute parity. For a discussion on RAID, see 1.4.4, “Failure and recovery” on page 39.

GPFS block size

The *GPFS block size* determines the minimum preferred increment for writing and reading files. It is also a contributing factor to the maximum size of a file.

11. The 256 KB write is broken into packets at the VSD layer (if max_IP_msg_size=60KB, four 60 KB packets and one 16 KB packet will result). When the GPFS makes read or write requests, the requests are sent to a server that owns the physical connection to the disk. This is done with the IP layer of AIX in the same way as transport protocols. Fragmentation and reassembly functions are required when read and write request sizes are greater than the size of a single IP packet.

max_IP_msg_size

This parameter defines the largest size of the packets that the IBM Virtual Shared Disk software will send between the client and the server. Its values can vary between 512 and 65024 bytes (63.5 KB).

12. Five 256-byte mbufs are acquired to contain VSD and IP headers (3). The mbuf management facility controls two pools of buffers: A pool of small buffers, which are simply called mbufs, and a pool of large buffers (which are usually called *mbuf clusters* or just clusters). 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 virtual memories. This means that they always reside in physical memory and are never paged out.

mbufs

An *mbuf* is a kernel data structure used in processing IP packets. Each 256 byte mbuf consists of a 28 byte header and a 228 byte data area. If more than one mbuf is required for an IP datagram, the mbuf header can contain a pointer to an *mbuf cluster* (also known as mcluster), which is usually 4096 bytes, or the datagram can be put into a linked list of mbufs, each pointing to its own mbuf cluster.

13. A CSS send pool(spool) buffer equal to the data of each packet is acquired. Five additional 256-byte mbufs are acquired to keep track of the buffer. In total, the 256 KB write uses 256 KB of GPFS page pool space, 10 256-byte mbufs, and 256 KB of CSS send pool space.

spool

The *spool*, short for CSS0 send pool, is an allocation of memory that is effectively a staging area for information to be sent over the switch. Likewise, the rpool is the CSS0 receive pool for receiving information sent over the switch.

Both GPFS and VSD have hard dependency on a high-speed interconnect between all SP nodes (SP Switch). The effective functioning of the switch requires that the rpool and spool settings for the switch be set to their maximum values of 16 MB each.

14. VSD copies data from the GPFS page pool buffer into the send pool buffers. At this point, the application's data has been copied twice, once into a GPFS page pool buffer and a second time from the page pool to the CSS send pool.
15. The VSD client sends the five IP packets it created to the VSD server via the IP switch.
16. The mbufs and CSS send pool space are held only until the data is transferred onto the switch; however, the request buffer and pagepool buffer are held until the write is completed by the server and an acknowledgment is received.

Request blocks

This parameter limits the number of virtual shared disk requests from a specific client node that can be pending at a given time. This includes requests to both local and remote devices.

17. Once the transferred data is received by the VSD server CSS receive pool, the CSS driver forward each packet to the VSD through the IP layer of AIX.

ipqmaxlen

Ipqmaxlen controls the number of incoming packets that can exist on the IP interrupt queue. Since both GPFS and the Virtual Shared Disk use IP, the default of 128 is often insufficient.

18. CSS allocates two mbufs and one receive pool buffer (64 KB) for each VSD packet that makes up a single request.

rpool

The *rpool* is the CSS0 receive pool for receiving information sent over the switch.

Both GPFS and VSD have hard dependency on a high-speed interconnect between all SP nodes (SP Switch). The effective functioning of the switch requires that the rpool and spool settings for the switch be set to their maximum values of 16 MB each.

19. Once all packets of a request have been received by the VSD server, a buddy buffer is allocated if it is available, and data is reassembled into it. If the buddy buffer is not immediately available, the request is queued and the data remains in the CSS receive pool.

buddy buffer

The *buddy buffer* is used by the IBM Virtual Shared Disk on the server to handle disk I/Os. The VSD server uses buddy buffers for temporarily storing data for I/O operations originating at a client node and to handle requests that are greater than the ip_message_size. Buddy buffer can be tuned by changing the minimum and the maximum buffer size allocated to a single request. These values can be set by VSD perspective or with the `vsdnode` command.

The SP Switch adapter is always used for interfacing Virtual Shared Disks. It is, therefore, recommended that settings of 4096 (4 KB) and the maximum file system block size the VSD node is serving are chosen for minimum and maximum buddy buffer sizes.

A buddy buffer is the buffer used by the IBM Virtual Shared Disk on the server to handle disk I/Os. The VSD server uses buddy buffers for temporarily storing data for I/O operations originating at a client node and to handle requests that are greater than the ip_message_size.

20. The VSD server releases all mbufs and CSS receive pool space associated with the request if the buddy buffer is allocated.

    VSD calls the Logical Volume Manager (LVM) strategy routing to schedule the disk write through the device driver.

    Every Virtual Shared Disk device has a logical volume defined and configured in the system. Thus, every VSD I/O request eventually becomes an I/O request to the associated logical volume. The VSD software layer transparently handles the mapping of VSD device I/O requests to the associated logical volume I/O requests.

21. The device driver performs the write. The driver waits momentarily in an attempt to write a data block of size *max_coalesce* or larger. On RAID systems, the max_coalesce should equal the RAID stripe size.

max_coalesce

max_coalesce is a parameter of the SSA device driver, which can be critical when using RAID. It allows the device driver to coalesce requests that have been broken up to satisfy LVM requirements. This is required for effective RAID performance for writes. 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.

queue_depth

This parameter specifies the maximum number of commands that the SSA disk device driver dispatches for a single drive for an hdisk.

22. VSD releases both the buddy buffer and the write request buffer after the completion of the write by the Logical Volume Manager (LVM) drivers.
23. VSD acquires a mbuf header to send a completion respond to the VSD client.
24. The VSD client releases the request block and drives GPFS completion processing.
25. GPFS completion make the page pool buffer available for use by another application call.

1.2.2 Read data flow

Figure 3 on page 13 shows the flow of a read system call. The read processing flows through the same points as GPFS write processing, but the data movement is opposite, and the stress points (such as rpoolsize, spoolsize, request blocks, buddy buffers) are different. For simplicity, we made the assumption that this is the first read of a one block file.

We define only those tuneables that are specific to reads only. Please refer to 1.2.1, “Write data flow” on page 4 for a description of the various tuneables that are pertinent to both writes and reads (such as rpool size, spool size, request blocks, buddy buffers).

The read flow begins with a read system call issued by the application. The following steps explain the read flow on GPFS.

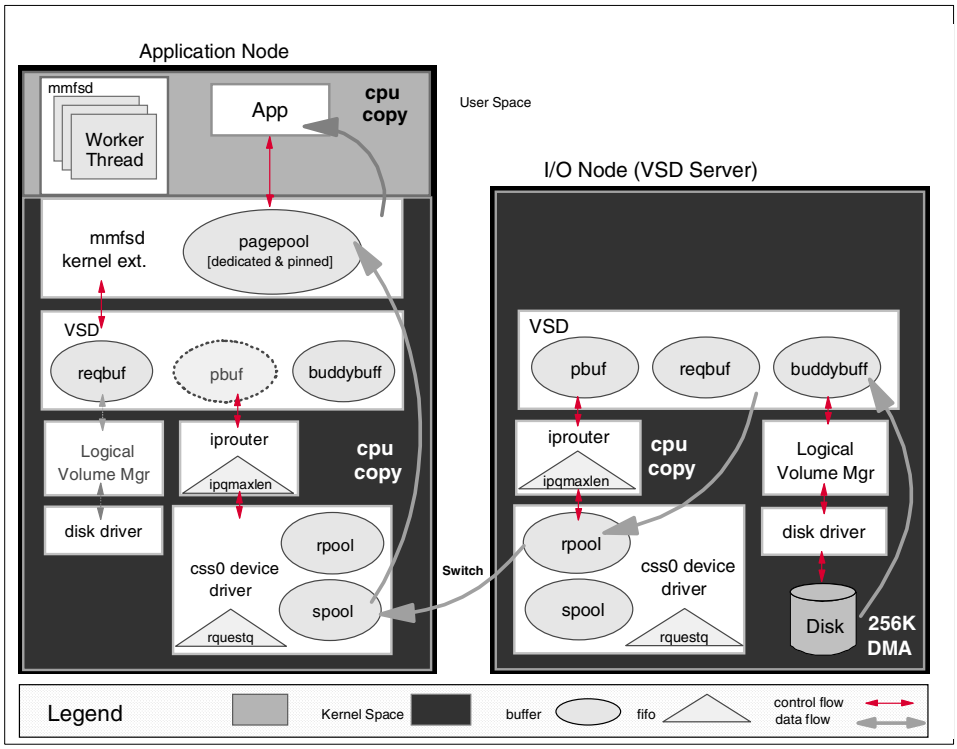


Figure 3. Control and data flow for a read operation

1. The application makes a read call with a pointer to a buffer in its space.
2. GPFS checks if the required data is already in the GPFS page pool or not. If so, GPFS will move the data from the pagepool into the application buffer and returns control to the application. If not, continue with the next step.
3. A buffer for the data to be read is first acquired with the necessary tokens required to preserve the consistency of the data. If there is no buffer available, a buffer is made available by writing the oldest buffer out to disk.
4. A GPFS daemon checks the necessary locks. If a read token is already held, skip to step 7. The mmfsd requests a read access token from the Token Manager Server. The request contains the required-byte-range and the desired-byte-range for the read.
5. The Token Manager Server determines if any conflicts exist for the read request. Assuming there is no exclusive access token currently held for the file, the Token Manager Server will grant a read token for a byte range

determined by the Token Manager Server (see 1.4.3, “Consistency” on page 39).

6. The mmfsd receives the Token Manager Server response containing the read token.
7. GPFS schedules a prefetchThread. The prefetchThread calls VSD strategy routine, which obtains an mbuf used to construct a request packet containing the VSD and IP headers.

prefetchThreads

The *prefetchThreads* parameter controls the maximum possible number of threads dedicated to prefetching data for files that are read sequentially. The actual degree of parallelism for prefetching is determined dynamically by the daemon.

8. IP is called to send the request over the switch to the VSD server since the amount of space required is much smaller than the write case.
9. Read request arrives at the VSD server.
10. CSS programs forward the request packet through the IP layer of AIX to VSD.
11. CSS allocates an mbuf to hold the VSD packet.
12. The VSD request is copied into a VSD request buffer.
13. The mbuf header is freed.
14. The VSD server allocates a buddy buffer and a read request buffer.
15. The VSD server constructs a disk read request in the pbuf.

pbufs

The *pbufs* are control structures used at the VSD server to describe each read or write request that is pending. Each IBM Virtual shared disk, regardless of its activity and regardless of whether the node is a client or server, will be allocated the same number of *pbufs* on a node.

Each puf is 128 bytes long. The pbuf shortage affects overall performance by causing physical disk operations. However, it does not imply that increasing the number of pbufs will improve performance because requests will be queued at other parts of the system.

16. The LVM strategy routine is invoked to schedule the read through the disk driver.

17. Disk data read into buddy buffer.
18. VSD takes control after the read is completed.
19. The VSD server sends the data to the client through the IP layer across the switch. As in the case of clients sending large requests to the server on write requests, the VSD server fragments the read data into multiple packets that get sent to the client.
20. VSD allocates ten mbufs and 256 KB of CSS send pool space.
21. VSD copies data from the buddy buffer to the CSS send pool buffers.
22. The VSD server forwards five packets through the IP layer to be sent across the switch to the client.
23. The VSD client node CSS driver receives an interrupt as the data arrives at the receive pool and forwards the packets to the VSD client. One rpool 64 K buffer and two 256 byte mbufs are used for each arriving packet.
24. The VSD server buddy buffer, read/write request buffer, and request buffer are released at this point.
25. On the completion path, the requested data flows back to the client into the receive pool from the switch.
26. After receiving the data, the VSD completion process is invoked, which copies the data to the GPFS buffer.
27. Return control to GPFS.
28. The GPFS thread moves the required data to the application buffer and returns control to the application.

1.2.3 Potential bottlenecks on writes and reads

Several factors may significantly hinder GPFS write and/or read performance. We discuss these items with the help of Figure 4 on page 16.

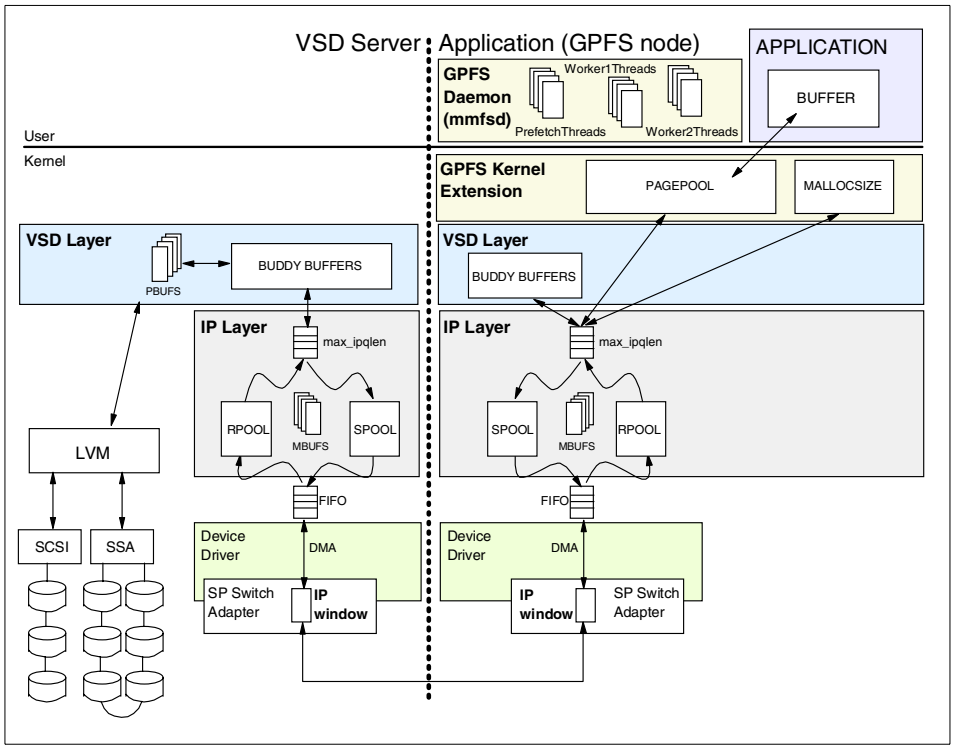


Figure 4. Potential bottlenecks to GPFS data flow.

As the diagram shows, a number of potential bottlenecks are involved in reads and writes. Incorrect settings for these key items will strongly affect performance.

- Insufficient pagepool capacity

  If the mmfsd daemon is unable to allocate buffers from the pagepool, GPFS may begin thrashing as it attempts to free up space.

- CPU Available on Token Manager Server

  If the Token Manager Server is unable to respond to token requests in a speedy manner, the read will be blocked.

- Insufficient mbuf capacity on VSD Client

  If the IP layer is unable to allocate mbufs, all GPFS and other IP communications will be blocked.

- Insufficient buddy buffer capacity on VSD server

If the VSD server is unable to allocate buddy buffers, the server will not be able to respond to VSD client requests. After two seconds, the VSD client will assume a failure and retransmit the write or read request.

- Insufficient mbuf capacity on VSD Server

  If the IP layer is unable to allocate mbufs, the server will not be able to respond to VSD client requests. After two seconds, the VSD client will assume a failure and retransmit the write or read request.

1.3 GPFS software structure and required services

This section describes both those software subsystems that are included within GPFS and any external software subsystem on which GPFS depends.

Internal subsystems and components

We begin by introducing the file system metadata components and five internal subsystems: (1) internal data structures; (2) the configuration manager, (3) the stripe group manager, (4) the metanode, (5) and the token manager server. The last four functions are implemented within the GPFS daemon, *mmfsd*, as different services or *personalities*. A single GPFS daemon may provide more than one service. Every node that is part of the GPFS domain runs mmfsd.

External subsystems

In understanding the operation of a GPFS environment, it is important to have at least a basic understanding of the external infrastructure that supports the GPFS architecture. We conclude this section with an examination of some external subsystems utilized by GPFS. This can be broken down into four different but dependent areas: (1) the IBM Virtual Shared Disks and IBM Recoverable Virtual Shared Disks, (2) the SP Switch environment, (3) the High Availability Subsystem or RS/6000 Cluster Technology, (4) the System Data Repository (SDR).

1.3.1 Internal data structures

Internally, GPFS is implemented by striping user data across multiple disks on multiple storage nodes. Distributed protocols coordinate metadata updates from multiple nodes.

The metadata architecture utilized by GPFS has similarities to the BSD fast file system[2].

[2] McKusick, Marshall K., William N. Joy, Samuel J. Leffler, and Robert S. Fabry, *A Fast File System for UNIX*, 1983

Inodes

Structures called *inodes* contain an array of disk addresses of data blocks for the file as well as owner, striping, and replication status information.

Indirect blocks

Indirect blocks contain additional disk addresses for data blocks for files too large to be represented in a single inode.

Directories

Directories are files with information on how the file namespace is organized. The namespace is a tree: Each directory is a file whose contents map file names with inodes. GPFS directories are sparse files; their contents may be addressed in units called *directory blocks*. A hashing scheme is used to find the directory block for a given file name. The capacity sizing of a GPFS file system must take into account the space needed for metadata structures, such as directories, inodes, and indirect inodes.

Data allocation map

The *data allocation map* keeps track of which disk blocks are in use and which blocks are available for data.

Inode allocation map

The *inode allocation map* keeps track of which disk blocks are in use and which blocks are available for inodes.

Stripe group descriptor

Similar to a BSD superblock, a *stripe group descriptor* holds a concise description of the file system. It contains the current information about about a stripe group and its components including the release of the file system under which it was created, read and write quorum sizes, striping and replication flags, and so on. Each of these structures are kept on disk and some are also cached to improve performance for metadata operations.

ACL file

GPFS supports *access control lists* (ACLs). ACLs are used to provide finer grain access control than ordinary UNIX file permissions. All ACL information is stored in an ACL file.

1.3.2 Configuration Manager

The *Configuration Manager* has overall responsibility for the correct operation for the nodes that impact GPFS in some way. In particular, it selects which node is to act as the Stripe Group Manager for each file system and appoints a successor node should one of those nodes fail. It also determines whether a *quorum* exists, which, in turn, is the number of application nodes necessary for file system usage to continue. This prevents separate uncoordinated updates of critical metadata, and their consequences, in the event of a machine network partition. For GPFS file systems, a quorum is 50 percent plus 1.

If the node fails whether the Configuration Manager itself is running, it will be replaced by another Configuration Manager with the aid of the Group Services subsystem. The node selected for the Configuration Manager is the first one listed in the mmfs group, which is dependent on the order in which the nodes came up. Group Services is discussed in 1.4.4, “Failure and recovery” on page 39. The CPU consumption of the configuration manager is minimum.

1.3.3 Stripe Group Manager

Each GPFS file system will have one and only one *Stripe Group Manager* at a time. The Stripe Group Manager, sometimes called the *File System Manager*, is responsible for maintaining availability information for the various disks that make up the file system. It processes various changes, such as adding or removing disks, changing disk availability, and repairing the file system. When data migration is warranted, the Stripe Group Manager initiates and coordinates the data migration. Mount and unmount processing of file systems is performed on both the stripe group manager and the node requesting the service.

A *stripe group* is related to a *file system*. A stripe group is a collection of disks that make up physical storage. For instance, one may speak of a particular stripe group’s block size. A file system is an administrative resource that may be mounted. For instance, one may speak of accessing the /gpfs1 file system.

The Stripe Group Manager plays an important part in write performance: It controls which regions of disks are allocated to each node, therefore, allowing effective parallel allocation of space for all nodes using the file system. Application nodes request allocation of space in accordance with the application’s needs. The Stripe Group Manager allocates more than the immediate requirement so that each new write will not require a message from the Stripe Group Manager.

In order for the Stripe Group Manager to perform recovery, handling of node failures, and online configuration of disks, a set of special purpose log files are maintained. The Stripe Group Manager determines which nodes are assigned which log files. With the use of log files, the Stripe Group Manager is able to clean up after a failure of a node on which the stripe group was mounted.

1.3.4 Metanode

When an application requests read or write access to a file, GPFS first determines if the file is already open. If it does, the node that opened the file will have certain pertinent metadata cached in a *Metanode* including the original access (read-only, write-only, read and write access). The Metanode may be thought of as a metadata manager; it manages all directory block updates.

The location for the metanode may change. For example, if a node gets access to a file, it may become the metanode.

1.3.5 Token Manager Server

The *Token Manager Server* synchronizes concurrent access to files and ensures consistency among caches. The granularity of locking may be whole files or portions of files. In addition, the Token Manager Server also performs some synchronization for GPFS internal data structures associated with allocation and file metadata. The item being accessed (for example, a file) is termed a lock *object*. The per-object lock information is termed a *token*. Locking is implemented as a single Token Manager Server per file system, plus one or more Token Managers, which behave as clients to the Token Manager Server.

A token needs to be invalidated when another node requests a lock that conflicts with the existing token. The token manager mediates the migration of tokens. This process is termed t*oken stealing*.

There is one Token Manager Server per file system located at the Stripe Group Manager node. If the Stripe Group Manager is moved to another node, the Token Manager Server moves with it.

1.3.6 VSD

IBM *Virtual Shared Disk* (VSD) is a subsystem that enables nodes in one SP system partition to share disks with the other nodes in the same system partition. It is also known as a logical volume that can be accessed not only

from the node it belongs to, but also from any other node in the system partition.

A VSD node can be a VSD server or a VSD client. A VSD server is a node that owns a number of VSDs. It reads and writes data to VSDs as requested by other nodes. However, VSD clients are nodes that request remote access to VSDs. It should be noted that a node can be both a VSD server and a VSD client node at the same time.

VSDs can be managed by the graphical user interface of PSSP that helps to perform shred disk management tasks without having to remember the commands. You can also run virtual shared disk commands from the SMIT panels.

IBM Virtual Shared Disk states can also be changed from the command lines. Figure 5 on page 22 shows the command that moves Virtual Shared Disks from one state to another.

All the VSDs information is stored in the SDR in the SP control workstation, and they can be viewed and changed by using the Virtual Shared Disk Perspective graphical user interface, SMIT interface, or the command line interface.

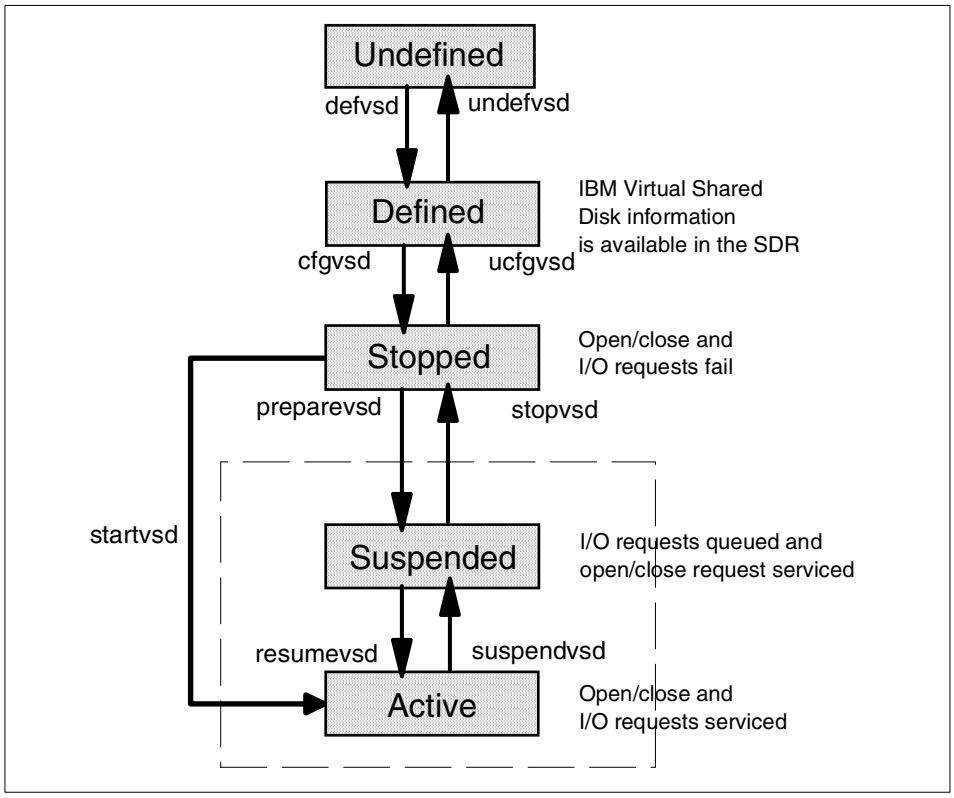


Figure 5. VSD states and associated commands

VSD architecture

Figure 6 on page 23 shows how VSD gains access to remote disks within the SP system.

Each disk in an SP is actively served by only one node and can only be accessed directly by that node. Furthermore, remote node disks are accessed through the switch network.

The routing is done by the IBM Virtual Shared Disk device driver that interacts with the AIX logical Volume Manager (LVM). The device driver is loaded as a kernel extension on each node. Thus, raw logical volumes can be made globally accessible.

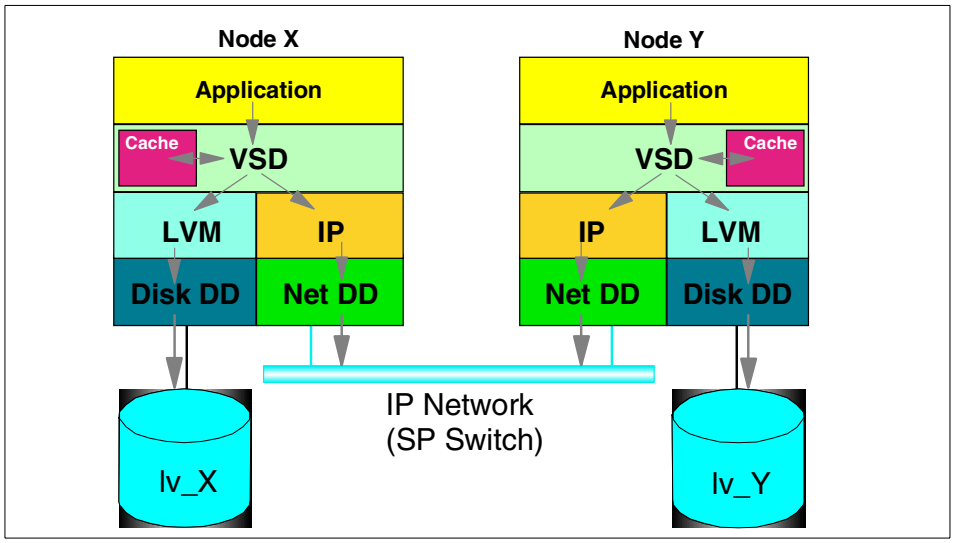


Figure 6. VSD architecture

The following four figures show the progression for write and read flow through the VSD layer. The steps are indicated by arrows with numbers. The first step is an application call into the GPFS daemon (mmfsd).

1.3.7 Recoverable Virtual Shared Disks

Recoverable Virtual Shared Disk is a program that is used with VSDs to provide high availability for Virtual Shared Disks (VSD) against node failure by subscribing to Group Services. When a node fails, RVSD is informed by Group Services.

Recoverable Virtual Shared Disk (RVSD) recovery subsystems, rvsd and hc, respond to changes in the status of the system by running recovery scripts and notifying client applications. The subsystems operate as daemons named rvsdd and hcd. They use the Group Services component of PSSP.

The rvsd subsystem

The rvsd subsystem controls recovery for the RVSD component of PSSP. It invokes the recovery scripts whenever there is a change in the group membership. The `ha.vsd` command controls the rvsd subsystem. When a node goes down, a disk, a disk adapter, switch adapter, or cable fails (in general, anything that is in the path to the disk or node), the rvsd subsystem notifies all surviving processes in the remaining virtual shared disk nodes so that they can begin recovery. If a node fails, recovery involves switching the

ownership of any disk served by this node to a secondary node. If a disk adapter or cable fails, recovery involves switching the server node for a volume group to the secondary node.

When the failed component comes back up, recovery involves switching resource ownership back to the primary node.

The rvsd subsystem uses the notion of *quorum*, the majority of the Virtual Shared Disk nodes, to cope with communication failures. If the nodes in a system partition are divided by a network failure, so that the nodes in one group cannot communicate with the nodes in the other group, rvsd uses quorum to decide which group continues operating and which group is deactivated.

Figure 7 shows a simple system with one twin-tailed recoverable virtual shared disk configuration. RVSD is being used to protect both VSD servers.

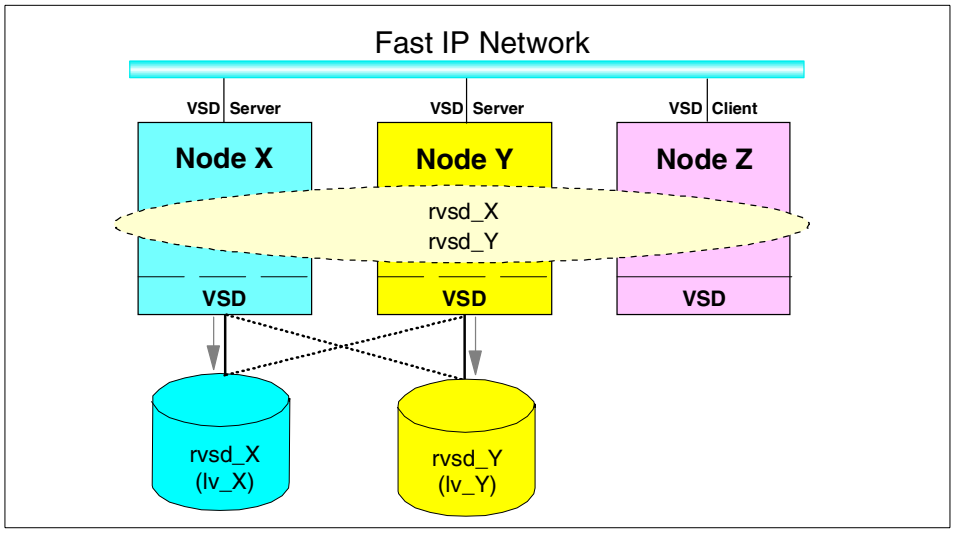


Figure 7. Recoverable VSD

In case node X fails, RVSD will have the VSD secondary server, node Y, take over the disk subsystems from the primary node and become the server for those VSDs while the primary node is unavailable as shown in Figure 8.

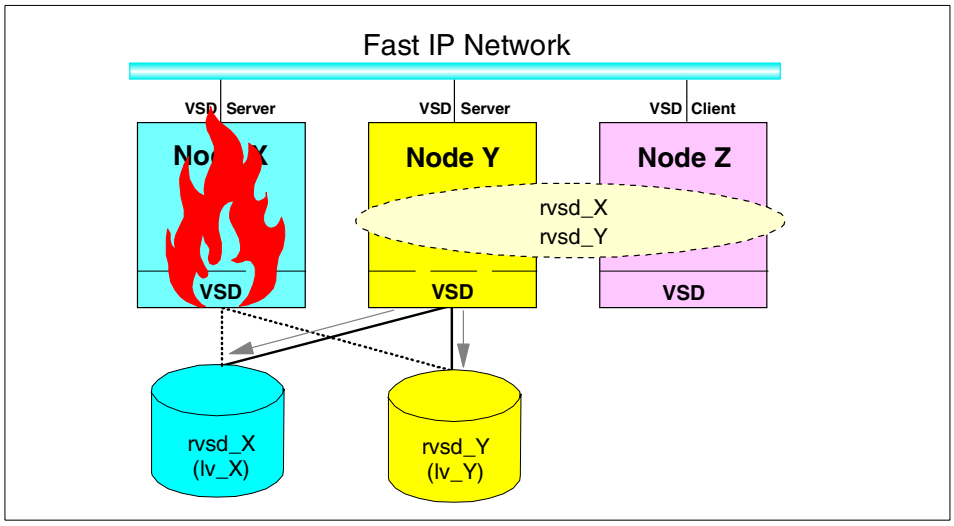


Figure 8. VSD take over

The RVSD concept is to allow not only one node (VSD server primary) to have access to a set of VSDs, but also a second node (VSD server secondary) in case one of the following fails:

- VSD server primary node
- Switch adapter
- Disk adapter
- Disk or network cable

Recovery is transparent to applications that have been enabled for recovery. There is no disruption of service. However, the amount of time required to failover to a secondary server depends on the number of volume groups that must be varied online.

In 1.4.4, “Failure and recovery” on page 39, we describe how Recoverable Virtual Shared Disk (RVSD) interacts with GPFS.

1.3.8 SP Switch

The correct planning and installation of switch hardware and its associated environment is beyond the scope of this book, yet its complexity and importance should not be overlooked. For information relating to this, you can refer to *Understanding and Using the SP Switch,* SG24-5161.

This section, however, provides an overview to other areas that are important in the operation of an SP Switch environment and the architecture that is set up to interact with other RS/6000 SP subsystems.

The configuration of the SP Switch can best be summarized in Figure 9. Note the similarity in initialization processes for both the primary and non-primary nodes. It is not until the switch is started from the control workstation that a node begins performing primary node functions.

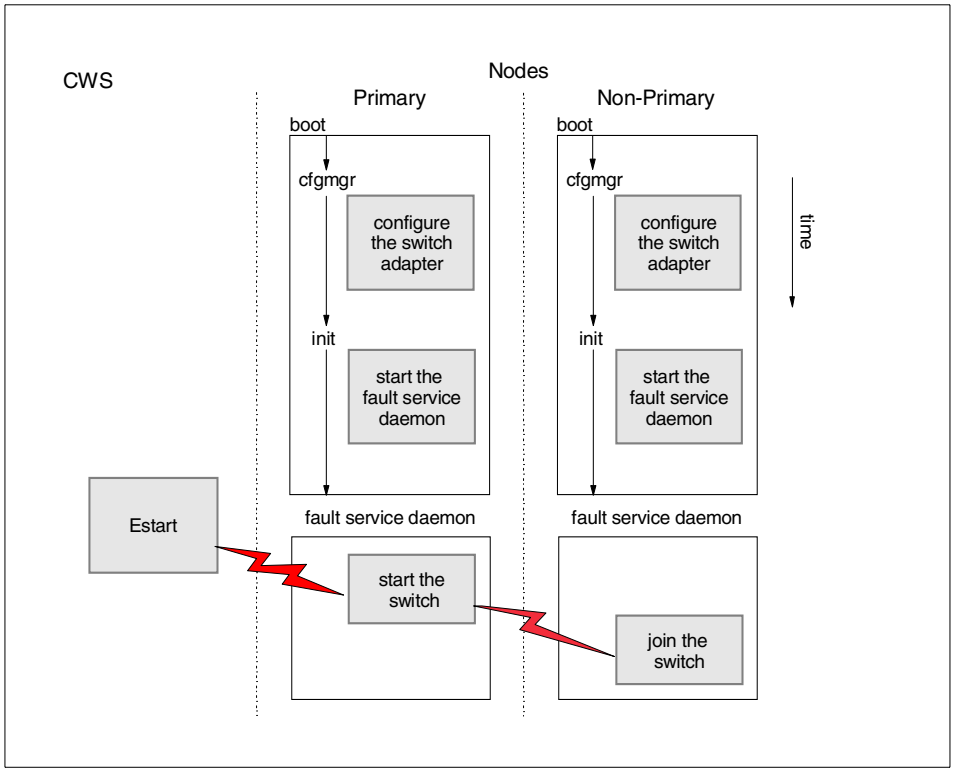


Figure 9. Switch initialization process

Configuration of the SP Switch adapter

For the most part, the configuration of the SP Switch adapter occurs during a node's boot process. The first part is performed by AIX in detecting the hardware adapter. Having successfully detected the new device, the program cfgmgr continues the process by creating and configuring the device through the execution of the command `cfgtb3`. The configuration process performs the following steps:

- Removes any conflicting definitions then defining the css0 device within ODM.
- Confirm the physical location of the adapter and rectifies any bus conflicts. Note that conflict resolution is not applicable to nodes that use SP Switch MX adapters or SP Switch PCI adapters.
- Creates the special file, that is, the device, /dev/css0.
- Loads the cssdd3 device driver and sets the device to being available. In a correctly operating system, this can be confirmed on all nodes by using the `lsdev` command as shown in the following example:

```
[serv6:/]# dsh -av 'lsdev -C -l css0' | dshbak -c
HOSTS -------------------------------------------------------------------------
v06n01i            v06n03            v06n05            v06n07
v06n09             v06n11            v06n13            v06n15
-------------------------------------------------------------------------------
css0 Available 00-f1000000 SP Switch Communications Adapter (Type 6-A)
```

- Loads the adapter firmware. This is, of course, dependent on the adapter type. From the previous `lsdev` output, the Type information indicates the adapter type versus the firmware loaded as shown in Table 2. You can also execute `lsdev -P -c adapter | grep Switch` to see a list of supported switch types. Note that the 6-F type is applicable for only the IBM S-70 systems and not RS/6000 SP systems.

Table 2. Switch adapter type and corresponding firmware information.

Type (from `lsdev` command)	Adapter Type	Firmware file loaded
6-9	MCA	xilinx_file3
6-A	MX	xilinx_file3mx
6-F	PCI	xilinx_file3pci

- Sets the device as being available in the ODM.
- Loads and starts the CSS kernel extension called fault_service_SP.
- Executes the adapter's POST diagnostics. This puts the *Trail Blazer Interface Chip* (TBIC) in reset after the diagnostics are complete, thereby, disabling the adapter. The TBIC is the part of the SP Switch adapter that manages the connection to the switch board from the node.

If the SP Switch adapter initialization process completes correctly, the adapter's adapter_status in the ODM is set to css_ready as demonstrated in the following screen:

```
[serv6:/]# dsh -a 'odmget -q"name=css0 AND attribute=adapter_status" CuAt' | dshbak -c
HOSTS -------------------------------------------------------------------------
v06n01i            v06n03            v06n05            v06n07
v06n09             v06n11            v06n13            v06n15
-------------------------------------------------------------------------------

CuAt:
        name = "css0"
        attribute = "adapter_status"
        value = "css_ready"
        type = "R"
        generic = "D"
        rep = "s"
        nls_index = 10
```

Starting the SP Switch daemon

The next phase of the SP Switch initialization process is to start the fault service daemons on each node in the RS/6000 SP system. This daemon is responsible for initializing and monitoring the switch and, in fact, operates slightly different depending on the role of the nodes it is on. First, these roles will be discussed; and second, the process the daemon goes through will be explained.

In an RS/6000 SP environment, a node can have one of three roles in relation to managing the SP Switch: Primary, primary backup, or secondary. By default, most nodes are defined as secondary nodes if they are not the primary or primary backup node. This means they have no special role in managing the switch.

Alternatively, one node in an SP partition will be nominated the primary, and a different node will become the primary backup. The primary node is responsible for starting, managing, and monitoring the SP Switch's operation. This includes managing recovery from node, link, and chip failures.

The primary backup node, as the name would suggest, monitors the primary node for its failure. If the primary backup node determines that the primary is no longer reachable, then it will perform a primary node takeover and start performing the primary's tasks.

With respect to these roles, the fault service daemon, fault_service_Worm_RTG_SP, performs three main services:

- During the daemon's initialization, or whenever SP Switch network topology changes, it calculates the routes between nodes using the Route Table Generation code. This part of the daemon is performed by all nodes.

- The daemon, during switch initialization, runs the Worm code, which determines the current usable switch topology. Unlike the route calculations, this daemon service is performed only on the primary node.
- The daemon is also responsible for reporting and servicing Error/Status packets to and from the switch fabric. It also implements relevant recovery procedures for node, chip, and link failures. Again, only the primary node receives the service packets from the switch fabric; however, all nodes will deal with adapter faults and communication with the primary node.

With the functionality of the daemon now explained, the next step is to explain its operation during its start up. This is handled by the /usr/lpp/ssp/css/rc.switch script. This script is executed by the init process during the nodes boot phase. The script will actually perform the following steps:

- Back up old log files.
- Kill any fault service daemons already running.
- Set the switch_responds attribute for the switch_responds class for that node to zero.
- Extracts from the ODM the type of adapter used and creates symbolic links to the relevant libraries for that adapter type.
- From the Switch_partition class, it checks if the node is a primary or primary backup node. If it is one of these, then it updates the SDR to indicate that there is no longer a primary or primary backup node.
- Retrieve the ODM attribute adapter_status and updates it to the node's adapter_config_status in the switch_responds class in the SDR. If this attribute is not set in the ODM, or is no set to css_ready, the script exits with an error.
- The adapters' IP address, netmask, whether it uses ARP, switch node number, switch number, and switch chip are all extracted from the ODM using different attributes associated with the css object. This is shown in the following screen capture:

```
[serv6:/]# dsh -w v06n01i 'odmget -q"name=css" CuAt | egrep "attrib|value"'
v06n01i:        attribute = "switch_node_num"
v06n01i:        value = "0"
v06n01i:        attribute = "switch_number"
v06n01i:        value = "1"
v06n01i:        attribute = "switch_chip"
v06n01i:        value = "5"
v06n01i:        attribute = "switch_chip_por"
v06n01i:        value = "3"
v06n01i:        attribute = "arp_enabled"
v06n01i:        value = "yes"
v06n01i:        attribute = "netaddr"
v06n01i:        value = "129.40.32.81"
v06n01i:        attribute = "netmask"
v06n01i:        value = "255.255.255.0"
v06n01i:        attribute = "state"
v06n01i:        value = "up"
```

- The attributes shown previously are then used to configure the css0 interface, leaving the interface in a down state, and then the fault service daemon is started with arguments of the extracted switch parameters given.
- Finally, just prior to exiting the script, a program called usconfig is run. This utility predefines some User Space window parameters.

At the time that fault service daemon is executed, a number of things are performed by it before normal activity starts. It starts the adapter's microcode, and it takes the TBIC out of reset. This enables the adapter.

After this, the fault service daemon starts its waiting phase. It waits for activities, such as:

- Service packets from the primary node.
- Service packets from the switch fabric if it is the primary node.
- Interrupts from the adapter.
- Management requests from E-commands, such as `Estart`.
- Locally occurring errors, which each node handles itself.

At this point, the fault service daemon is now prepared for the SP Switch to be started.

Starting the SP Switch

The third and final phase of initializing the SP Switch is the execution of the `Estart` command. This command is executed on the control workstation, however, most of the startup processes actually occurs on the primary node.

Before this occurs, the issued `Estart` command performs a handful of tests from the control workstation to check that the system is correctly configure.

- It checks if the switch fabric has lost its clock source. This is done by performing a SDRGetObjects on the clock_change attribute in the switch class. This is set to `yes` whenever there is a change in clock-related hardware variables. This can be reset using the `Eclock` command; however, a typical `Estart` error is shown in the following screen capture:

```
[serv6:/]# SDRGetObjects Switch clock_change
clock_change
yes
[serv6:/]# Estart
Estart: 0028-070 Cannot Estart, the clock source for one or more
of the switch boards has changed.  Eclock needs to be run to re-establish the
clock distribution of the switch clock network.
```

- It verifies that both the oncoming primary and oncoming backup node are up.
- On these two nodes, it also checks if the fault service daemon is running on them.
- It checks that the oncoming primary node is not isolated. If the oncoming primary backup is isolated, however, a warning message is displayed, and another node will be selected as the primary backup.
- If all four checks are passed, then the Estart program starts the Estart_sw script on the oncoming primary node.

Now that the Estart process on the control workstation has transferred control to the Estart_sw program on the system's oncoming primary node, the points that follow will explain what occurs during the remainder of this phase.

- The first step is the distribution of the partition's topology file from the oncoming primary node. These are found in /spdata/sys1/syspar_configs/topologies. The file expected.top, if it exists, will be used for debugging switch problems and should not exist after switch testing is complete. The name of the file that is used can be extracted from the SDR.

  The oncoming primary node then checks how many nodes have switch adapters against the number of successful topology file distributions in the previous attempt. This value is stored in the SDR as the attribute num_nodes_success as shown in the following screen output. If these do not match, then the file is sent to all nodes.

```
[serv6:/]# SDRGetObjects Switch_partition topology_filename
topology_filename
expected.top.annotated.1
[serv6:/]# SDRGetObjects Switch_partition num_nodes_success
num_nodes_success
          8
```

- The next step is to verify that the oncoming primary node is *not* already the primary node. If it is, the `Estart_sw` changes the state of the fault service daemon to make it a secondary node (described in “Starting the SP Switch daemon” on page 28). This change also occurs to the node acting as the primary backup node to avoid conflicts problems between a primary node takeover and the current `Estart` script.
- After performing the primary node check, the fault service daemon on the oncoming primary node is signalled so that it can start the switch. The `Estart_sw` script will then wait for the daemon to finish switch initialization before continuing. This is considered complete when a file called act.top.*pid* exists in /var/adm/SPlogs/css, where *pid* is the process ID number of `Estart_sw`. Note that this file eventually becomes topology.data within the same directory

  From this log file (sample shown in following screen capture), the `Estart_sw` script updates the SDR with information about the primary node name and the primary backup node name also shown in the following screen capture.

```
[v06n09:/]# cat /var/adm/SPlogs/css/topology.data
Number of active node(s) seen by the Worm:
8
Number_of_linksbad: 0
The primary backup node is:
14
The following switch node(s) are active:
8
14
12
10
6
4
2
0
The topology file used by the Worm:
/etc/SP/expected.top.annotated.1
[v06n09:/]# SDRGetObjects Switch_partition primary_name primary_backup_name
primary_name primary_backup_name
v06n09       v06n15
```

- Finally, unless `Estart_sw` encounters an error through the fault service daemon not initializing the switch, the SP Switch environment is ready for normal operation. This includes updating the SDR switch_responds class by setting the switch_responds and autojoin attributes to one for each node that successfully joined.

  At the same time, all the fault service daemons enable both the IP and User Space protocols for their nodes. For further information about how IP integrates with the switch, please refer to“IP interface to the SP Switch adapter” on page 33

IP interface to the SP Switch adapter

The initial development of the SP Switch adapter did not use IP to provide communications over the SP Switch. This situation changed as more and more applications that already used TCP/IP or UDP/IP to communicate began operating in an RS/6000 SP environment. By making IP and its associated protocols available on the SP Switch adapter, the nodes could also communicate with machines not in the RS/6000 SP system.

For an RS/6000 SP, the IP protocol is managed by an AIX kernel extension, which interacts with various adapters on the node. In particular, for the SP Switch adapter, the if_ls interface layer is responsible for communications with the adapter hardware. This is shown in Figure 10.

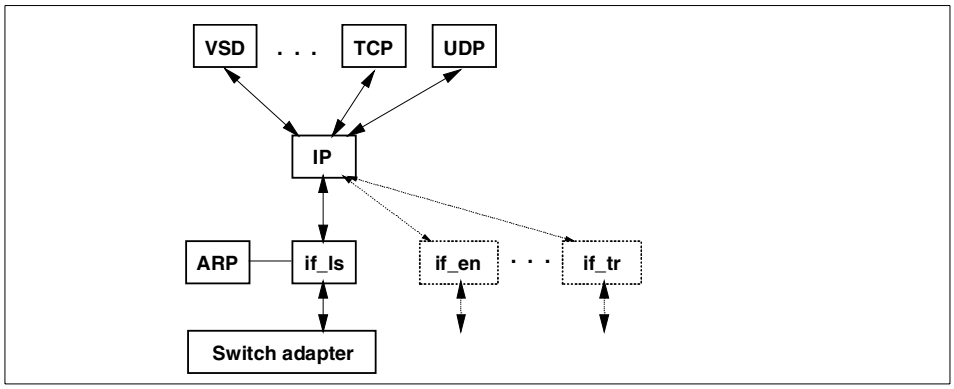


Figure 10. IP to switch interaction

As you can also see from this figure, as well as the traditional support for TCP and UDP, nodes use a variant of UDP to implement the communications of the IBM Virtual Shared Disk subsystem. The sending and receiving of data through IP and the switch adapter is performed using a similar processes. In both instances, the processes varies slightly for different packet sizes. For sending packets, if the packet is small enough (less than 228 bytes), a single

kernel mbuf is allocated, and the data is copied to it. For larger packets, one or more data structures called mclusters are also allocated. For IP communications, the kernel allocates the mclusters directly from the switch adapter's send pool space. The mcluster sizes can be 4, 8, 16, 32, or 64 KB, and is allocated based on availability and size. That is, the smallest block that can hold all the data is allocated. The mbuf that is still allocated now references the mcluster, or for even larger data requirements, multiple mbuf/mcluster combinations are allocated with the original mbuf referencing the first mbuf/mcluster pair in a linked list fashion as shown in Figure 11.

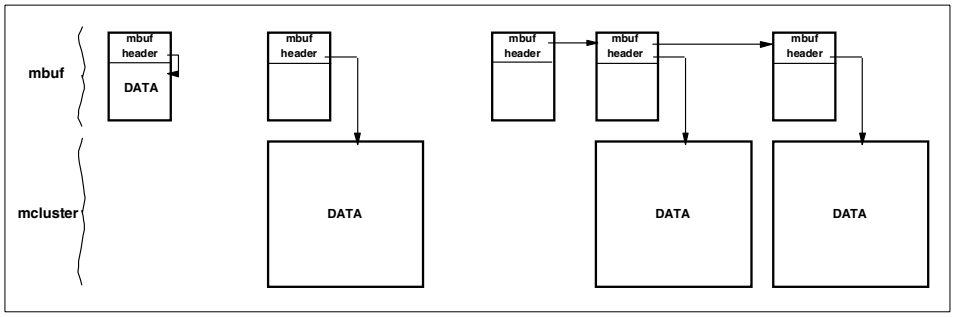


Figure 11. Kernel structures for IP over switch communications

Once prepared, the interface layer receives an mbuf structure indicating where all, if any, the mclusters containing data are. It then creates an entry in the send FIFO queue. Because the send FIFO is a shared memory queue, the switch adapter is able to read and update the queue. From the FIFO record, it detects if there is more data in the send pool and starts transmitting the data into the switch. Once complete, the send pool space is freed for other packets.

For reading, effectively the reverse occurs. For both small (less than 228 bytes) and large IP packets, an entry is made in the receive FIFO when a packet is received from the switch. For small packets, this also contains the data. For large packets, relevantly sized mclusters are allocated from the receive pool in the same size and manner as the send pool as mentioned earlier. When the last packet is received, the shared receive FIFO is updated with information that points to the new receive pool data.

Following that, the interface layer is notified via an interrupt. Once interrupted, the interface layer processes requests on the receive FIFO and then passes the recreated mbuf information to the IP layer for normal processing.

This is a simple explanation of how the SP Switch adapter interacts with the IP network protocol. For more detailed information relating to the management of pool space and the emulation of broadcast packets for IP over the switch adapter, please refer to *Understanding and Using the SP Switch,* SG24-5161.

1.3.9 Clustering subsystems

The RS/6000 Cluster Technology (RSCT) is a set of dependent technologies, which currently include Topology Services, Group Services, and Event Management.

- Topology Services is the building block for RSCT. It is a subsystem that is distributed across all nodes in a SP environment. It maintains availability information for all the nodes and adapters, which is used to provide the Network Connectivity Table. The Network Connectivity Table is applicable for both the Ethernet and SP Switch adapters for that SP domain. This information is then utilized by Group Services' Reliable Messaging library.
- The Group Services subsystem is a fault tolerant, highly available, and partition-sensitive service that provides a means for monitoring and coordinating changes to the situation of another subsystem operating on a RS/6000 SP partition. Group Services achieves this through a number of sub-modules.
  - Client Control Module - This module accepts, manages, and handles requests from Group Services clients, such as IBM Recoverable Virtual Shared Disk and Event Management.
  - Meta Group Control Module - Meta groups are collections of Group Services daemons that support various user groups. This modules manages the behavior of the daemons.
  - Name Server Module - This module manages the Group Services namespace.
  - Reliable Messaging Module - By utilizing this module and the Network Connectivity Table, Group Services is able to provide reliable, sequenced delivery of messages to Group Services daemons.
  - Topology Services Client Module - Group Services utilizes this module to access the services offered by Topology Services.

  Through these modules, Group Services provides services grouped by node partitions. The Group Services daemon operates on each node within a partition and once on the control workstation for each partition. For IBM Virtual Shared Disk servers and GPFS nodes, Group Services

maintains membership information that can be viewed for any node as the following example demonstrates:

```
[serv6:/]# dsh -w v06n01i 'lssrc -ls hags' | dshbak
HOST: v06n01i
-------------
Subsystem         Group            PID     Status
 hags             hags             17574   active
 4 locally-connected clients.  Their PIDs:
 18586 14208 25192 26960
 HA Group Services domain information:
 Domain established by node 0.
 Number of groups known locally: 3
                    Number of   Number of local
 Group name         providers   providers/subscribers
 cssMembership            8           1           1
 ha_em_peers              9           1           0
 ha.vsd                   8           1           0
```

As you can see from the final three lines, there are three group services that this IBM Virtual Shared Disk server has subscribed to along with seven or eight others. In order, the services are the switch adapter group, the Event Management group, and the IBM Virtual Shared Disk group. Also, the process IDs of the daemons that contribute to these groups are listed.

Similarly, a GPFS node subscribes to these groups as well as two other GPFS-only groups.

```
[serv6:/]# dsh -w v06n09 'lssrc -ls hags' | dshbak
HOST: v06n09
------------
Subsystem         Group            PID     Status
 hags             hags             17838   active
 5 locally-connected clients.  Their PIDs:
 17582 15054 18962 17214 23340
 HA Group Services domain information:
 Domain established by node 0.
 Number of groups known locally: 5
                    Number of   Number of local
 Group name         providers   providers/subscribers
 Gpfs.1                   4           1           0
 GpfsRec.1                4           1           0
 cssMembership            8           1           2
 ha_em_peers              9           1           0
 ha.vsd                   8           1           0
```

Note the two extra GPFS services, of which, only four nodes have joined. This is as the system was configured. Also, note the extra processed ID is listed, which is for the GPFS daemon, mmfsd.

- The third subsystem, Event Management, provides information to the other subsystems by monitoring various software and hardware resources within the system. This includes resources, such as: Disks, filesystems, CPUs, and processes. Like Group Services, it achieves this through the use of three sub-components. These are:
  - Event Manager daemon - This ties the Event Management subsystem together. It handles the collection of resource information as they are reported by resource monitors, registers event requests from clients and notifies them when a relevant event occurs.
  - Resource Monitors - These are programs that monitor specific system resources and report the results to the Event Management daemon.
  - Event Management Clients - These are also programs that can perform three tasks. First, they can register an event expression with the Event Management daemon (for example, when the CPU is seventy percent busy), receive information when the event occurs (for example, the current CPU usage), and query resources that the Event Manager stores (for example, the top ten CPU using processes).

1.3.10 System Data Repository

The System Data Repository (SDR) is the location where SP configuration data is stored. It is stored in one of two ways: As a class, which is made up of attributes and objects, or as a file. There are two main areas where the GPFS architecture interacts with the SDR.

- The first is GPFS itself. It stores two text files in the SDR, which are accessed by GPFS nodes upon their startup. These two files are mmsdrfs and mmsdrcfg1. To view their contents at any time, you can extract copies by using the `SDRRetrieveFile` command.

  The mmsdrfs stores GPFS cluster and file system information. The cluster information is used to determine which nodes are available to be part of the GPFS quorum. Locally, this information is also stored in /etc/cluster.nodes. The file system information is transposed into the nodes local filesystem file. An interesting note here is the vsddisk entries. These are used by GPFS for internal processing and are not part of the file system stanza. They are made of up to five records on a line with each record describing the IBM Virtual Shared Disk name, its size, and failure group.

  The mmsdrcfg1 file holds information pertaining to the operation of GPFS. This includes settings for items such as, pagepool, mallocsize, priority, and so on. When the GPFS daemon on a node starts, this file is retrieved from the SDR and is stored locally as /var/mmfs/etc/mmfs.cfg.

- The second use of the SDR is for IBM Virtual Shared Disk information. It stores VSD operational parameters in the SDR object node. These are best extracted using the `vsdatalst` command with the `-n` option. Actual VSD configuration information is also stored in the SDR in the VSD_Table and VSD_Global_Volume_Group objects. You can again use the `vsdatalst` command with the `-v` and `-g` options to see these details, respectively.

  The screen output that follows is a sample of each command. It has been edited to show only the first few lines for conciseness.

```
[serv6:/]# vsdatalst -n
       VSD Node Information
Initial Maximum     VSD      rw      Buddy Buffer
  node               VSD      IP packet   cache   cache request request minimum maximum size: #
number host_name     adapter    size    buffers buffers   count   count    size    size maxbufs
------ ------------ -------- --------- ------- ------- ------- ------- ------- ------- -------
     1 v06n01i       css0         61440      64     256     256      48    4096  262144      33

[serv6:/]# vsdatalst -v
              VSD Table
VSD name              logical volume  Global Volume Group          minor# option    size_in_MB
--------------------- --------------- ---------------------------- ------ --------- ----------
gpfs10n3              lvgpfs10n3      gpfs77n3                         11 nocache    8672

[serv6:/]# vsdatalst -g
      VSD Global Volume Group Information
                                                 Server Node Numbers
Global Volume Group name        Local VG name   primary      backup eio_recovery     recovery
------------------------------- --------------- -------      ------ ------------     --------
gpfs10n5                        gpfs10                5         0                0          0
```

Of course the SDR is used elsewhere by the RS/6000 SP system. However, in relation to the operation of GPFS, the above two areas store information relevant to the architecture of GPFS.

1.4 GPFS operation

This section describes how various tasks are performed by GPFS. Special attention is paid to those aspects that permit GPFS to run on multiple nodes and multiple disks concurrently.

1.4.1 User interfaces

All file systems provide a number of data management services, such as the ability to duplicate files, remove files, rename files, and so forth. As a POSIX file system, GPFS appears the same as any other POSIX file system but with exceptional capacity and read (or write) performance. The user performs ordinary file system operations, such as copying a file, and so forth, on GPFS in the same manner as they would on other standard file systems, such as a Journaled File System (JFS) or Network File System (NFS).

1.4.2 Security

File permissions utilize standard AIX user and group security mechanisms. GPFS implements standard AIX file access controls through its use of directories.

GPFS and VSD use `sysctl` commands. Such commands are made secure through Kerberos and ACL files.

1.4.3 Consistency

Since all nodes have access to all disks, there must be coordination of which node has control of which data. GPFS consistency is managed by a centralized Token Server Manager for each file system. Tokens may be viewed as locks that may be passed around among nodes in an SP. Whenever a GPFS component desires access to a lockable object, it requests access from the Token Server Manager for that file system. Objects that are controlled by tokens include portions of files, inodes, allocation maps, and other metadata. GPFS supports the following lock modes for byte-range lockable objects (files):

- Not-Locked
- Read-Only
- Exclusive-Write (conflicts with all other locks)

When multiple nodes of a parallel job open a file with write access, the first task is given a Exclusive lock for bytes 0 to infinity. When the second task requests write access, both will be granted a Exclusive lock, but the associated byte range will be split among the two tasks. For instance, if the first task is writing bytes 0-1023, and the second task is writing bytes 1024-2047, the first task will be granted a write token for bytes 0-1023, and the second task will be granted a token for bytes 1024 to infinity when it first accesses byte 1024.

Shared resources in multi-threaded parts of GPFS use locks to insure proper synchronization.

Token management can be CPU and memory intensive. An appropriate node should be selected to be the Token Manager Server. The Token Manager Server is always located on the same node as the Stripe Group Manager.

1.4.4 Failure and recovery

Normal operation of GPFS depends on several factors:

- All nodes must be accessible via the switch.

The requirement that all nodes must be accessible via the switch comes from token management and metanode requirements. If communications across the switch fail from a node, mmfsd terminates normally on the node so that it will not degrade the system.

GPFS utilizes Group Services, RS/6000 Cluster Technology, to report the failure of nodes or the switch. If Group Services reports that the switch or network has failed, applications performing file operations will fail, and the error will be set to ENOTREADY or ESTALE.

It is also possible for TCP/IP to report an error. When this happens, GPFS waits five minutes for Group Services to report the error. If Group Services does report the problem, processing proceeds as described in the preceding paragraph. If Group Services does not report an error, the error is presumed to be on this node, and mmfsd does a cleanup and a forced exit.

- The mmfsd is operating on every node.

  When the mmfsd daemon fails, it is fenced off from using GPFS to insure that a consistent state is reached within the file system. If the failure was software, the SRC (mmfsd is started by the SRC) reaps the SIGCHLD and immediately restarts the mmfsd daemon.

  Whenever a node boots, it retrieves configuration information from the SDR. The mmfsd daemon goes through a four phase process where votes take place to ensure all nodes are recovering in step and correctly. If a second failure occurs while recovery is proceeding, the voting cannot continue. When this happens, the mmfsrec process will take action based on the second failure.

- A retry protocol makes the VSD transport layer reliable.

  The VSD transport layer is built on an unreliable IP datagram transport protocol. The VSD layer recovers from lost or corrupted packets by an exponential back off retry algorithm. A request may be tried multiple times. Each time, the client waits twice as long as it did before the previous try.

- All disks make up the file systems are accessible.

  Loss of access will cause the file system to be forcibly unmounted.

GPFS is able to utilize logging, Group Services, RVSD, and high availability media technologies (disk mirroring, replication, and RAID) for recoverability and availability. These are described in the following sections.

1.4.4.1 Logging and GPFS

When a GPFS node (non VSD server node) fails, GPFS automatically tries to recover through the use of logs. Two copies of logs are kept in separate failure groups to assist with recovery. Although data that is in memory will be lost, the built-in recovery features ensure that data integrity is maintained within the file systems.

1.4.4.2 Group Services and GPFS

As previously mentioned, GPFS is designed to utilize Group Services part of the RS/6000 Cluster Technology (RSCT). Each GPFS node is a member in the high availability infrastructure. Each node runs the hcd daemon, and a membership list is maintained with the status of all nodes. These subsystems utilize a heartbeat to detect failures. User tunable values (interval, sensitivity) adjust the allowable time before a node is considered to have failed.

1.4.4.3 RVSD and GPFS

RVSD is normally used in conjunction with *twin-tailed* or loop cabling of the SSA disk subsystem between nodes to give high availability for volume groups and the associated VSDs. A twin-tailed configuration is one in which each disk is connected to two nodes: A primary and a secondary. In the event that the primary node fails, the secondary node takes ownership of the disks. It provides an alternate path to the disk subsystems from the VSD server nodes as shown in Figure 7 on page 24.

If the server node P fails, access to the data on all VSDs that it owns is lost. In order to avoid this situation, Recoverable Virtual Shared Disk (RVSD) and twin-tailed or loop cabling are implemented between Node P and Node S.

If you do not implement RVSD, and also do not have replicas, some disks in the stripe group will not be available when a node is down. In this case, GPFS will find that continuing the operation may compromise the integrity of the file system, and it will force the unmount of the file system so that no one can use it.

Normally, the VSD secondary node S does nothing with those VSDs that belongs to node P. In fact, it cannot access them since they are being held in use by the VSD node P, which is the primary.

RVSD provides protection against node failure by subscribing to Group Services. When node P fails, RVSD will be informed by Group Services. RVSD will have the VSD secondary server node S for each VSD perform the necessary functions to ensure all VSDs on the failed Node P can still be accessed by the clients.

This is achieved by having the VSD secondary server node S take over the ownership of the disk subsystem, vary-on the volume group containing rvsd_P, and make the VSD server node S become the server for those VSDs while the primary Node P is unavailable as shown in Figure 12 on page 43. Node S serves both rvsd_P and rvsd_S. Any I/O operation that was in progress, and new I/O operations against rvsd_P from GPFS, are suspended until failover is complete.

Thus, with RVSD, the disk subsystem becomes highly available since you can have continuous access to the VSDs even when the VSD primary server node P is down.

When node P is repaired and rebooted, RVSD switches the rvsd_P back to its primary, node P.

RVSD is a prerequisite for GPFS even when you do not care about the high availability of your file systems or do not plan to use any external disk subsystem. This is because GPFS needs some command in RVSD, for example, `fencevsd` and `unfencevsd`, that are necessary to ensure that the integrity of the GPFS file system will not be compromised.

A two-phased protocol, which uses several scripts (vsd.UP1, vsd.UP2, vsd.DOWN1, and vsd.DOWN2), is performed by the RVSD during failover (see Figure 12 on page 43).

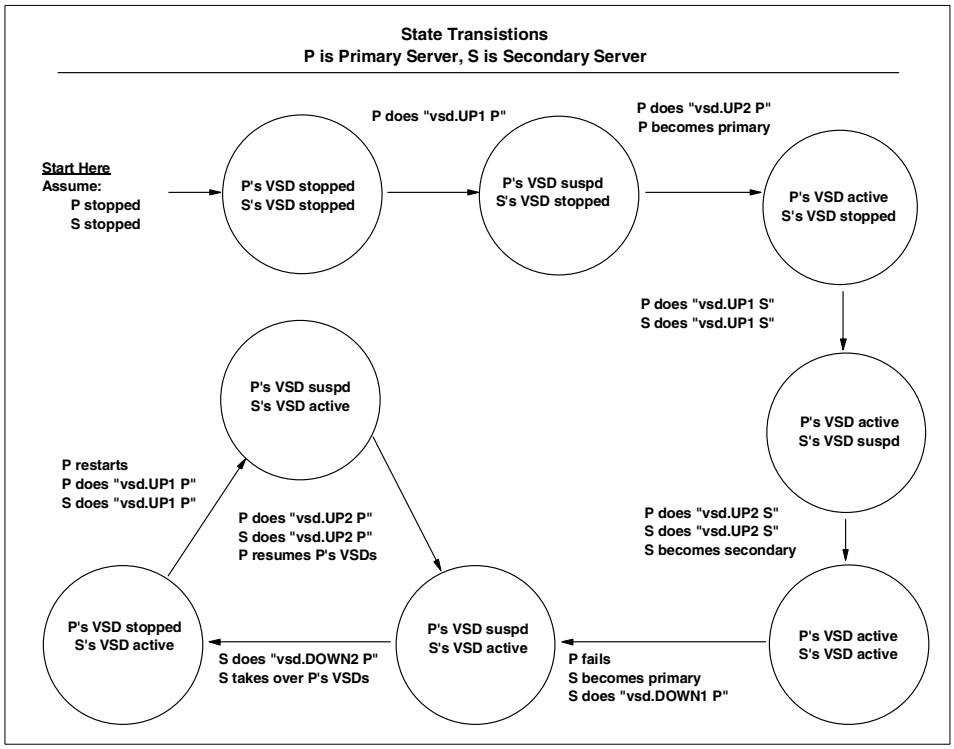


Figure 12. State transistions for RVSD recovery

1.4.4.4 Disk mirroring, replication, and RAID on GPFS

GPFS is able to utilize combinations of disk technologies designed to improve reliability.

Disk mirroring

The highest performance for increased availability is through AIX disk mirroring. This provides two or three copies of a logical volume. In the event of a failure, the other copy is used. Mirroring is performed at the lower layers of the I/O stack. Data is only sent through the switch once, but mirroring requires three times as many I/Os as non-mirrored disks for two-copy mirroring. If one VSD server goes down, availability to both mirrored copies is lost if not twin-tailed. Mirroring also includes additional cost for the added disk drives (100 percent additional cost for two-copy mirroring).

Disk replication

GPFS permits two copies of objects marked for replication. The copies may be established on a per file or per file system level (100 percent additional cost for only those files that are replicated). With replication, one copy is sent to one VSD server, and the other copy is sent to a second VSD server (assuming that the failure group is defined with a different VSD server). Therefore, communications is increased (travels over the switch twice), but the workload is easier on the VSD server (only one I/O performed per request). Replication is able to continue in the presence of a single VSD server failure.

RAID

The lowest performance disk availability enhancement, but probably the most inexpensive, is Redundant Arrays of Inexpensive Disks (*RAID*). A typical GPFS RAID configuration uses four data disks and one parity disk in a RAID-5 configuration. This means that each 256 KB write is spread evenly among the five disks along with 64 KB of parity information, which is also spread among the five disks (20 percent additional cost). If any single disk fails, the missing data can be reconstructed from the parity information spread out over the remaining disks. The parity information is automatically generated by the SSA adapter, but there is a performance penalty for writes. For instance, 6215 adapters (Campbell adapters) can normally stream at 35 MB/sec, but they slow to around 20 MB/sec when streaming to RAIDs. When one disk in a RAID set fails, the RAID set is said to be in *degraded mode*. Writes and reads to a degraded RAID set will incur an extra penalty. Finally, RAIDs are subject to *Read-Modify-Write* cycles. If a write is smaller than the RAID stripe-width (typically 256 KB), the adapter must calculate the parity for the entire stripe. To do this, it must read the unmodified portion of the stripe, calculate a parity based on the entire modified stripe, and write out the new portion of the block as well as the new parity. Read-Modify-Write cycles, therefore, perform several additional I/Os. For example, let us assume that we are using a 256 K GPFS blocksize and 7+p RAID. The RAID stripe width is 7 * 64K, or 448 K. The GPFS blocksize determines that GPFS will write information in increments of 256 K. Since the RAID stripe width is 182 K larger than the GPFS block size, this 182 K must be read in so that parity can be computed on the modified raid stripe. Then, the modified parity and the new data are written out. Read-modify-write cycles typically degrade performance in the range of two to three times slower.

1.5 Positioning GPFS and other file systems

This section gives a high level view of how GPFS compares against other popular types of file system. A summary is first given of several different file system types. The key advantages and limitations of GPFS, as compared to those file systems, is then drawn up in sections 1.5.2 and 1.5.3.

1.5.1 Comparison of GPFS with other file systems

There are several other file system types that are useful to compare GPFS against. These are:

- The AIX Journaled File System (JFS)
- Sun's Networked File System (NFS)
- The Open Software Foundations Distributed File system (DFS)
- The Andrew File System (AFS)
- IBMs Parallel I/O File System (PIOFS).

1.5.1.1 JFS

JFS is the default local file system used on IBM RS/6000 systems. JFS was developed according to the BSD model but with the addition of a JFS log used to track metadata changes. This log can significantly reduce the time needed to verify or repair a file system.

JFS is not a distributed file system. Without NFS, JFS file systems can only be accessed on the local nodes on which they have been configured. Because JFS is a local file system only, it cannot be directly used for parallel access from multiple nodes. Figure 13 demonstrates that, even in a networked environment, a JFS file system data can only be accessed serially.

Perhaps the main advantage that JFS offers over distributed file systems is that, because JFS operates on a local node only, it requires, by comparison, only a few system resources.

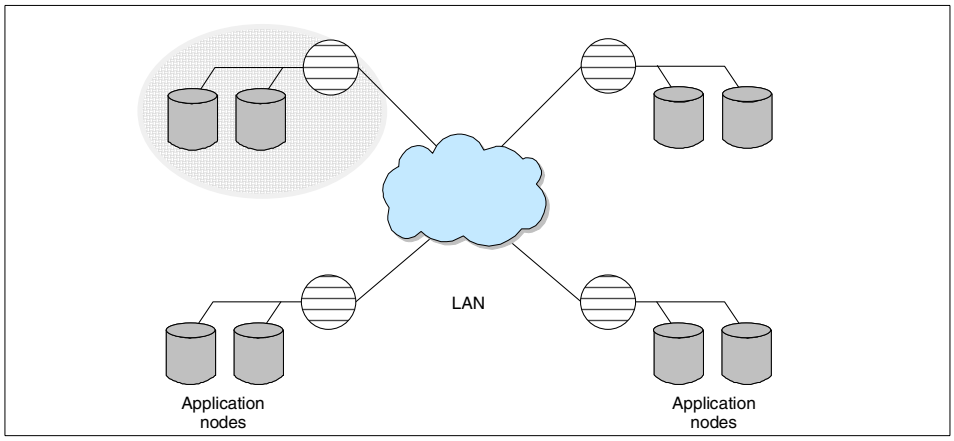


Figure 13. JFS offers local access to data only

1.5.1.2 NFS

NFS was initially developed by Sun Microsystems in 1984 and has been made available to other UNIX implementations. Today, it is undoubtedly the most frequently used method for making local UNIX file systems available to other nodes networked over TCP/IP. Remote nodes can access NFS served file systems for either read or write access according to how the servers have been configured.

NFS is the oldest distributed file system considered in this section. It is extremely convenient to utilize, as NFS support is integrated into most UNIX operating systems. NFS is very simple to set up when compared to the other distributed file systems.

NFS V2 was made available in 1985, and the NFS V3 protocol was published in 1995. Some of the advantages of NFS V3 over earlier releases are:

- Improved client write throughput
- Reduced server loading resulting in improved scalability and performance
- Increased support for ACLS
- NFS server support for large multi-gigabyte files

AIX V4.3 supports NFS V3. AIX also provides an NFS Version 2 client and server and is, therefore, backward compatible with an existing install base of NFS clients and servers.

Some of the drawbacks associated with NFS are:

Weak security
Historically, NFS is regarded as notoriously insecure. Although, today NFS file systems can be configured to use a simple Unix-style authentication mechanism, based on a list of trusted hosts, there is no built-in encrypted user authentication mechanism. However, NFS can be configured to use some of the securer authentication mechanisms, such as Kerberos or Diffie-Hellman. NFS authorization facilities include Unix style authorization, based on standard permission bits, or Access Control Lists. (ACLs). The implementation of ACLS on an NFS server requires the client to also support ACLs; otherwise, authorization falls back to the standard Unix style. Both NFS Version 2 and NFS Version 3 servers support ACLs by virtue of extensions that have been made outside of the NFS protocol.

Stale mount points
NFS has a tendency for client mount points to become stale. Frequently, this results in commands, such as `df`, hanging when they attempt to access remotely served file systems. Stale mount points are caused by the lack of synchronization between server and client. The most common scenario is for file systems to be unmounted on their server node, or for the server node to be rebooted without a prior dismount of the file-systems by their respective clients. Stale mount points can usually only be resolved by rebooting individual client systems.

Stale metadata
NFS clients view of file system metadata is prone to being stale. This is due to various inconsistencies that can develop between different clients' view of the file system.

Administration complexity
The ease with which NFS file systems can be served across the network means that it is all to easy for NFS file systems to become nested across multiple systems. The worst scenario is for application file system hierarchies to evolve into a complex chain of NFS file systems exported from multiple servers. The main reasons for this situation arising are lack of management discipline and/or poor application design. This type of chaining can soon become very difficult to manage and equally difficult to back track. NFS is best constrained so that served file trees are not straddled across multiple servers. This restriction does not apply to other distributed file systems.

Poor scalability
NFS does not scale very well for two reasons. First is the fact that when NFS mount points are nested, the number of mount points increases according to a square rule. This is illustrated in Figure 14.

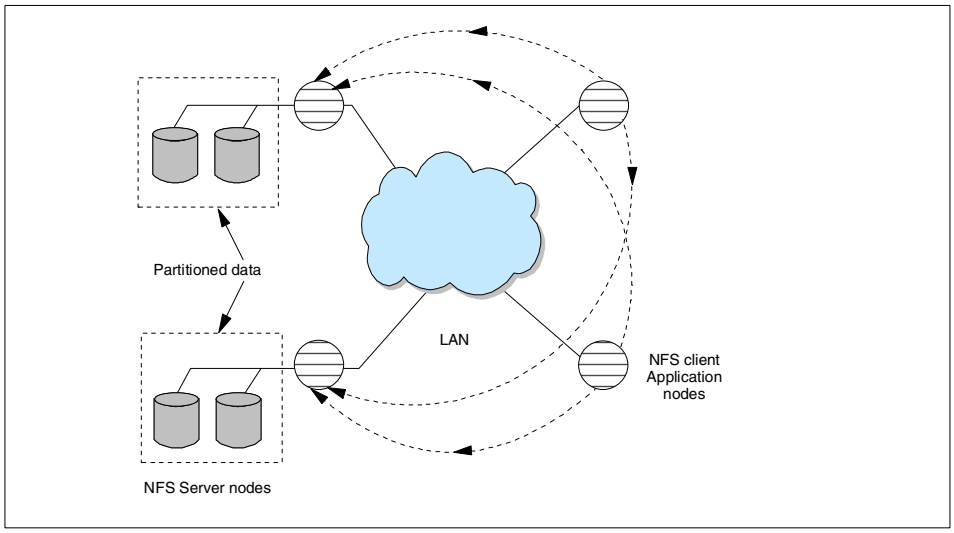


Figure 14. The complexity of NFS mount points

The second reason why NFS file systems do not scale very well is due to the reliability issues when NFS file systems are nested.

1.5.1.3 PIOFS

The IBM AIX Parallel I/O File System (PIOFS) was a predecessor to GPFS. Like GPFS, PIOFS was IBM proprietary software directed at the SP market only.

PIOFS was developed by IBM to provide fast and parallel access to large temporary files on an IBM RS/6000 SP over the SP Switch. Although GPFS is designed for the same operational environment, its use is not limited to temporary files only. PIOFS was directed primarily towards scientific and technical computing applications generating large amounts of temporary data. The maximum supported file size under PIOFS was a theoretical 128 TB, which was considerably larger than the maximum currently supported by GPFS.

PIOFS supports physical and logical partitioning of files. A file can be divided physically over multiple disks and servers and logically into multiple subfiles. These multiple subfiles enable data within a file to be viewed in different ways. These ways of viewing the files' data are termed subviews. Applications can use this facility via PIOFS API calls within C and FORTRAN environments. One limitation of PIOFS is that it does not support byte range locking.

PIOFS offered a good level of performance, and it was recommended for scratch storage. It was not designed to survive component failures. The product was withdrawn from marketing in December 1998.

1.5.1.4 DFS

DFS was developed as part of the Open Software Foundation's (OSF) Distributed Computing environment (DCE). DFS is the DCE component, or more correctly, the DCE application, that offers a distributed file system within the context of the DCE infrastructure. DFS has evolved to a large extent from the AFS file system.

DFS configuration is an order of magnitude more complex than any of the other alternative file systems described in this section because it requires the design and construction of a DCE cell and all of the associated DCE services.

DFS is a heterogeneous solution. It is not proprietary software and, hence, it is not an SP only solution. Unlike GPFS, DFS has not been specifically tuned to sequential access on large files. On the other hand, because GPFS has been tuned this way and integrated with IBM SP PSSP and VSD software, GPFS performance will be far better than DFS on the types of file and I/O patterns it has been specifically designed for.

DFS is really not comparable to GPFS although it has some of the key availability features of GPFS, such as replication. DFS is a distributed file system designed for read-mostly applications.

In DFS, any nominated file can be served by its own individual server. This is unlike NFS where servers operate at the file system or directory level only. DFS, therefore, makes it possible to optimize performance where accesses to specific individual files present an I/O bottleneck. On the other hand, it is not possible to stripe a single DFS server across multiple nodes. This is one of the key differences between DFS and GPFS. A GPFS file system can be striped across multiple disks and multiple nodes. It is easy to see, therefore, that a DFS server may soon become a performance and capacity bottleneck.

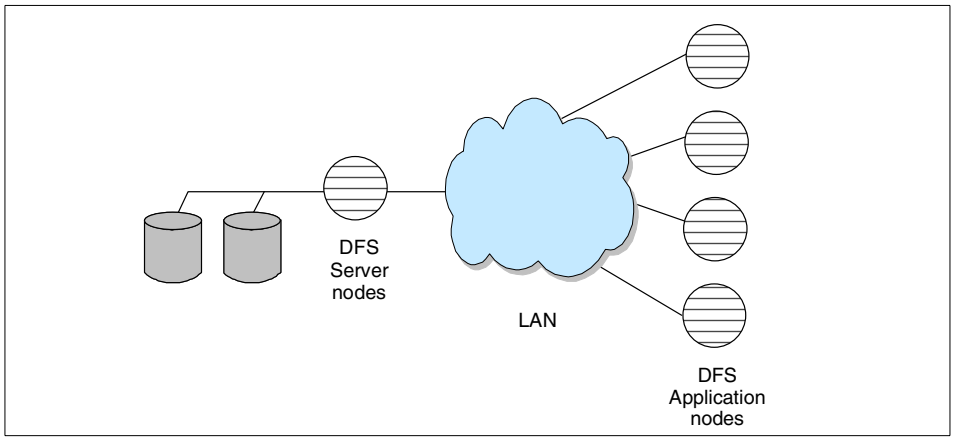


Figure 15. The DFS server bottleneck

1.5.1.5 Andrew File System

The Andrew File System (AFS) was initially developed at Carnegie Mellon University and is directed towards centralized management of a distributed file system across a campus type infrastructure.

One important advantage AFS has over NFS is that files can be accessed with transparency of their location. Clients access a file by name and do not need to know the address details of the node owning the disk on which the file data is physically stored.

AFS overcomes the main security problems posed by NFS in that it uses Kerberos for user authentication.

AFS supports advisory locking of an entire file but does not support byte range locking within a file. This limitation may impact both the portability of an application and the ease with which the applications data can be partitioned.

AFS and DFS are similar in architecture and offer many of the same benefits. As one component of DCE, DFS is the logical file system of choice requiring a tightly integrated DCE infrastructure for their applications. AFS represents a convenient stepping stone to organizations requiring a heterogeneous distributed file system without having to make a more profound commitment to a complete DCE infrastructure.

1.5.1.6 File system comparison summary

Table 3 summarizes the features of the various file systems discussed in this section.

Table 3. Comparison of file system features

Feature	JFS	NFS	AFS	DFS	PIOFS	GPFS
Scalability	N	N	N	Y*	Y	Y
Parallelism	N	N	N	N	Y	Y
Cross-platform	N	Y	Y	Y	N	N
Replication	N	N	Y	Y*	N	Y
Large files/file-systems	N	By partitioning only			Y	Y
Security	Y	N	Y	Y	Y	Y
Failure Recovery	N/A	N	Y	Y	N	Y
Centralized administration	N/A	N	N	N	Y	Y
Byte range locking	Y	Y	N	Y	N	Y
Physical file system	Y	N	N	N	N	Y

* Read via replicas

1.5.2 GPFS advantages

This section summarizes the key advantages of GPFS as compared to the file systems considered in 1.5.1.

1.5.2.1 Scalability

One of the prime advantages GPFS offers over other file system types is its scalability.

GPFS file systems can be striped across multiple disks on multiple storage nodes. This makes it very easy to scale GPFS file systems in terms of both capacity and performance by adding additional nodes as VSD servers and additional disk adapters and disks on VSD server nodes. This is in contrast to NFS and DFS, which can only serve a file system from the viewpoint of a single server.

VSD servers do not have to be dedicated but can, in fact, be a GPFS node themselves. This means that client applications can take advantage of spare CPU time on VSD servers. The merits of doing this will, of course, depend on the application and its predicted resource requirements.

GPFS grants a transparent access to all the data contained in a GPFS file system. Once a GPFS node has mounted the file system, the entire data is visible regardless of how many VSD servers have been configured to store the data. This is in contrast to NFS, where each client node must mount served file systems to form an aggregate whole.

With GPFS, it is possible for disks and nodes to be added or deleted from a file system while the file system is mounted.

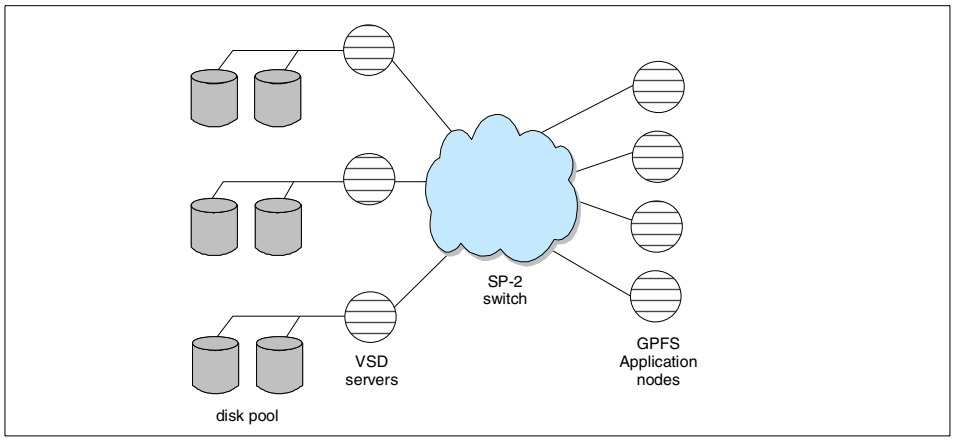


Figure 16. The scalability of GPFS

1.5.2.2 Performance

In terms of individual GPFS node to VSD server throughput, overall I/O performance for one node will, of course, not be as great as would I/O to a local JFS file system. Bigger block size will be better for GPFS performance. However, the first area where GPFS really scores in terms of performance is that the total aggregate I/O throughput for a multiple client application will normally be far in excess of other solutions, such as NFS and DFS. This is because the I/O bandwidth can be scaled across multiple VSD servers in order to satisfy the aggregate performance that is required.

The second performance advantage of GPFS is that, because it is a parallel file system, it allows multiple processes to access the same file simultaneously for read and/or write access from different nodes. PIOFS is the only other file system considered in this section that allows parallelism of applications in this way. Applications need to be specifically developed to take advantage of this parallelism, for example, by utilizing the MPI-I/O API. However, please note that while GPFS fully supports advisory byte range locking, PIOFS does not.

Another important performance consideration is that the write I/O of new data blocks is automatically balanced across all disks within the file system. This is due to GPFS' striping algorithm that functions transparently with use of the file system. Traditional local or distributed file systems are far more localized in terms of data placement, which greatly increases the risks of loading imbalances or performance bottlenecks.

Where it is efficient to do so, GPFS automatically invokes read prefetch and write behind algorithms. The algorithms are automatically invoked when GPFS detects that sequential I/O to a file is being performed. This greatly improves performance for applications that do intensive sequential read and write operations.

1.5.2.3 Capacity

GPFS permits file systems to be configured with much larger sizes than traditional file systems, such as JFS. As discussed in this section, these file systems are usually configured transparently across multiple disks and VSD servers. The precise maximum file system limits depend on a number of factors that the file system is configured with. For example, with a file system block size of 64 KB, an indirect block size of 16 KB and i-node size of 1 KB, the maximum file system size would be just under 70 GB. For the same file system, the maximum permitted file size would be 26.1 GB with replication.

There is an effective supported limit of 5 TB on the maximum file system for GPFS 1.2. File systems greater than 5 TB in size are not supported by IBM service. Note: This limit was 1 TB in GPFS V1.1.

For more information on the maximum permitted size for GPFS file systems, please refer to Chapter 2 of *GPFS: A Parallel File System*, SG24-5165.

1.5.2.4 Availability

GPFS supports three methods of extended data availability that should be considered according to individual requirements and cost restraints. These are:

- AIX LVM mirroring
- RAID-1 or RAID-5 disk arrays
- GPFS replication

GPFS replication facility permits multiple copies of a file or file system to be maintained automatically. The file system replication factor determines the number of copies it is required to store and may be set for individual files or for the entire file system. Replication is the simplest of the three options to

configure but has the greatest performance impact. Refer to 1.4.4, “Failure and recovery” on page 39 for details.

Like JFS, GPFS is also a logging file system. Logs are maintained for each GPFS node, which permit the fast recovery of data in the event of node failure. GPFS uses a token mechanism as an internal mechanism for insuring that data and metadata are consistent and correct.

GPFS supports an automount facility, which means that file systems can be configured to be mounted automatically when the GPFS daemons are successfully started. This increases the overall availability of the GPFS environment.

1.5.2.5 Simplified administration

GPFS administration is distributed in that a single administration command can be executed on one particular node so that it is also effective on all other GPFS nodes.

With GPFS, a single `mount` command makes it possible to gain access to the entire file system, no matter how many VSD servers it is configured over and whatever its size is. This makes it very easy to move the execution of applications to different nodes. This is in contrast to file systems, such as NFS, where a very large file system may be partitioned across several different server nodes. If there are n GPFS nodes in an application environment, then it takes only n mounts to make the GPFS file system available to all nodes no matter how many VSD servers support the file system. By contrast, if there are m NFS servers supporting a partitioned file system, it would take n*m NFS mounts to make the aggregate data available to all nodes.

1.5.3 GPFS limitations

1.5.3.1 Hardware environment

Today, GPFS is only supported within an IBM SP environment. File system communications between GPFS nodes and VSD servers functions are limited to the SP Switch only. This is because only the switch can deliver the bandwidth needed for a scalable parallel file system. Although it is possible to export GPFS file systems over NFS, GPFS is not a heterogeneous solution. This is in contrast to the NFS, DFS, and AFS file systems considered in this chapter.

1.5.3.2 Reliability issues

There are currently no flow control facilities in GPFS. An application issues its write requests to GPFS asynchronously, meaning that once the write has

copied the data to the GPFS pagepool, the application will continue processing as if the write had completed. In a sense, this is analogous to most UNIX file system I/O, as modified file system data is typically buffered in free memory until such time that it is physically flushed out to disk by the system's update daemon. However, the additional dependencies on the layers of VSD software increase the risk of reliability problems.

During the course of writing this redbook, it was discovered that if certain disks out perform others on a VSD server, the I/O requests to the slower disks may queue back up the VSD subsystem. Because these requests are normally issued on a round-robin or balanced random algorithm, a backlog is likely to form whenever there is a performance imbalance such as this and there is heavy application I/O. As the requests to the slower disk continue to queue up, a point may be reached where the VSD client attempts a repeat of the complete I/O request even though that request is still pending in the queue. This can result in an eventual time-out of the VSD sub-system and a device not ready error for the entire GPFS file system. (In the situation quoted above, the disks were, in fact, all compatible in terms of their specification but were different models of the same drive.) We recommend to configure GPFS file systems with disks of similar performance to avoid this problem.

1.5.3.3 Installation issues

An installation of VSD and GPFS software is not a turnkey process. The initial installation and configuration of VSD servers can be complex, as can an initial installation of GPFS. One particular trouble spot that often causes problems is the fact that it is necessary to set up a dummy IBM Virtual Shared Disk before a virgin GPFS configuration will complete successfully.

1.5.3.4 Performance constraints

The performance of GPFS depends greatly on specific application I/O patterns. Random access applications will, in general, require more overhead than sequential access because GPFS can not predict the access pattern and do prefetch. The application block size is also another issue. Applications performing intense small rewrites will not work efficiently under GPFS because updates require a read of the existing state of the remainder of the block.

Please refer to Chapter 2, "Application considerations" on page 57 for more details on GPFS and application considerations.

1.5.3.5 Memory mapped files

Memory mapped files are not currently supported on GPFS file systems.

1.5.3.6 File system primitives

The *stat()* function is not fully supported, and mtime, atime and ctime returned from the stat() system call may be updated slowly if the file has recently been updated on another node.

Chapter 2. Application considerations

Before GPFS can be properly sized or tuned for a specific environment, it is essential that the I/O behavior for the applications that make up that environment are fully understood. This chapter highlights the key application I/O issues for the GPFS implementor.

The degree to which the I/O behavior of applications can be analyzed and/or customized towards GPFS will depend on a quality set of documentation and/or access to source code or resource to a specific support department. When assessing the suitability of applications for which there is no source code, or legacy applications for which there is no sufficient technical documentation, it may be necessary to first undertake a detailed performance analysis to assess the applications key I/O characteristics.

2.1 GPFS application block Size

GPFS originated from a file system designed for multi-media, video-streaming applications. By its very nature, GPFS is, therefore, particularly appropriate for sequential file access within large files.

In GPFS 1.2, GPFS file systems can be configured with one of three block sizes. These are 16 KB, 64 KB, and 256 KB. Selection of an optimal block size for a GPFS file system depends on other factors than the application block size itself. However, it is easy to see that:

- Applications that have I/O buffers the same size as the GPFS block size will have an optimal block size in respect of its interface to GPFS.
- Applications and file systems that are both configured for 256 KB block sizes will outperform those configured for 16 KB and 64 KB block sizes.

Applications rewriting in buffer sizes smaller than the file system block size will incur an additional overhead than if they had written an entire file system block. This is because the new data must be merged into an existing file system block, therefore, requiring that entire block to first be retrieved from disk. Subsequent writes of the same block will not cause the block to be again reread from disk unless it has, in the interim, been modified by another node. This is true for rewrites that are not sector aligned. If the rewrite is sector aligned, then the data is written to the block, and the rest is mark as invalid.

Applications reading in buffer sizes smaller than the file system block size will, in fact, exhibit similar performance as if they had read an entire file

system block because of the read ahead performed by GPFS. In fact, performance will be slightly worse than this because of the CPU incurred in extracting and returning the smaller buffer to the application.

Many applications that are being considered for GPFS will be well suited to 25 KB application buffers. These are the type of applications based around large data file accessed with sequential read and writes. Some applications, however, may not be well suited to such large buffering. For these, there is the option of either 16 KB or 64 KB buffering. 16 KB might typically be used for random and small I/O buffering or for applications requiring access to a large number of relatively small files. A 16 KB setting optimizes the use of disk storage at the expense of data transfers while 64 KB offers a compromise between file system efficiency and speed of access. Whatever the final decision is from the application point of view, performance will be optimized if the GPFS file system block size is set to match the application buffer size.

2.2 GPFS application performance

It is recommended that, wherever possible, applications take advantage of the st_blksize member of the POSIX stat structure. This item returns the size of the file system. Applications can use this item to scale the size of their read and write buffers to match that of the host file system, thereby, ensuring portability if the application data is moved between different GPFS file systems. This argument is equally valid in the situation that portability is also required between JFS and GPFS file systems.

2.3 GPFS application I/O

This section reviews basic file and file system I/O patterns and how these can be exploited within GPFS.

2.3.1 Sequential and random application I/O

Application I/O may either be primarily sequential or random in nature. Sequential I/O involves the reading and writing of consecutive logical blocks, where the individual block size may either be fixed or variable. Random I/O involves the reading and writing of blocks from random points within a file. Again, the block size may either be fixed or variable. This is illustrated in Figure 17.

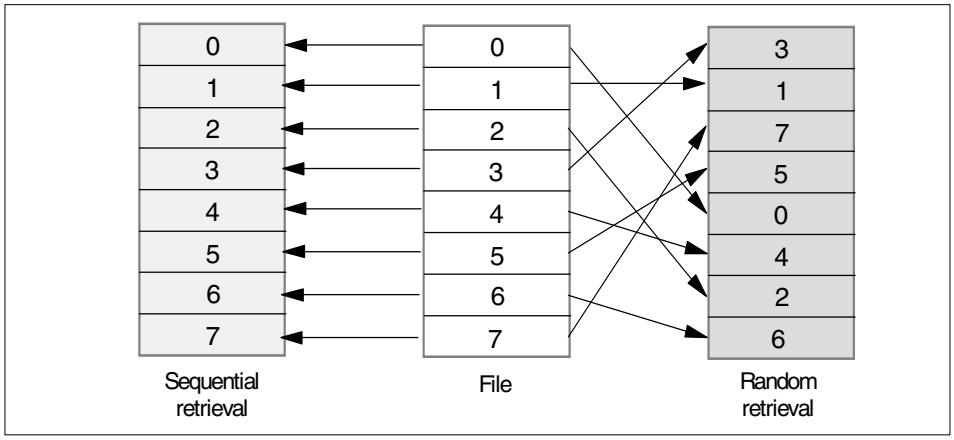


Figure 17. Comparing sequential and random I/O

There are important differences in the performance characteristics of sequential and random GPFS applications, which are considered in section 2.3.3.

2.3.2 Serial and parallel file system I/O

As well as the specific file access methods an application uses, I/O access can also be considered at the system level to be serial or parallel. Serial I/O relates to multiple nodes reading or writing to logically distinct sections of the same file or physically distinct files. This is a typical JFS type scenario. With parallel I/O, multiple nodes can access the same file, or the same regions of a file, simultaneously. This is a typical GPFS scenario. Figure 18 and Figure 19 illustrate the differences between serial and parallel I/O.

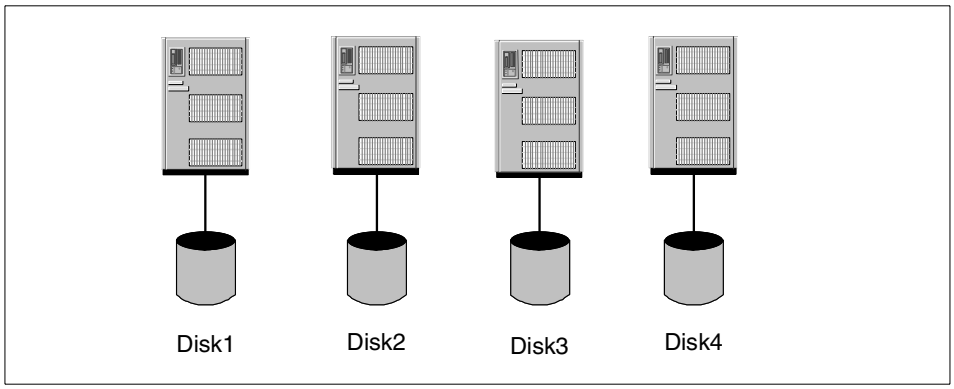


Figure 18. Comparing serial and parallel I/O

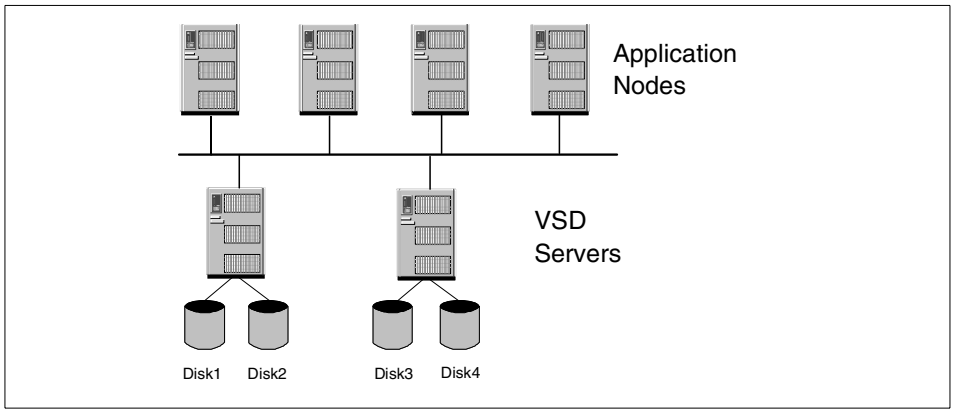


Figure 19. Parallel file system I/O

On a parallel file system, I/O can be further subdivided into different access patterns. These are parallel sequential, parallel random, and parallel strided. Refer to 2.4, “Data partitioning” on page 61 for further details on data partitioning.

2.3.3 Application I/O patterns and GPFS

Applications whose I/O is primarily random are not as well suited to the GPFS file system as those that do sequential I/O. Random reads on a file will function much slower than sequential because GPFS cannot know how to prefetch the data. This requires that the read I/O has to be synchronous, meaning the read cannot commence until the application specifically requests it.

Random writes to a file in buffers the size of the GPFS file system block size will see a performance similar to sequential writes as long as sufficient pagepool has been allocated. As with sequential writes of short buffers, short random writes will incur the same overhead in terms of the requirement to fetch an original file system block and merge into it the modified buffer.

In the tests that were performed during the course of writing this redbook, applications performing intense random I/O were seen to be extremely slow on the GPFS file system. GPFS may not be the best choice of file system for such applications. Refer to Chapter 6, “Test results” on page 199 for further details.

2.3.4 Exploiting GPFS read prefetch and write behind

GPFS has an internal prefetch algorithm that recognizes sequential I/O read patterns. Applications that make sequential accesses and, therefore, avoid seeks, get the best performance from GPFS. For strided I/O patterns, the use of MPI-I/O should be considered. This permits separate logical views of a file to be created and allows the separate logical views of the data to be separated from the physical disk access to retrieve the data. In this way, MPI-I/O can utilize parallel read operations to read along the logical views, thereby, taking advantage of the GPFS read prefetch buffering.

If the use of MPI-I/O is not desired, but it is required to perform a striding I/O pattern, there is another option. This is to open the file multiple times and to perform sequential reads from each file handle. The downside to this is that it is then up to the application to handle the coordination of read request to the separate file handles. Since the read ahead algorithm is based on file descriptors, having the same file opened multiple times makes GPFS think that the reads on each file descriptor are sequential; so, it turn on read prefetch.

Exactly the same considerations apply to sequential writes.

2.4 Data partitioning

For many parallel applications, data must be written from multiple tasks that are potentially on multiple nodes to secondary storage. *Data partitioning* refers to the mapping of bytes from memory to one or more files. The two primary questions that the application programmer faces when doing data partitioning are how to define the intra-file mapping and whether or not multiple files should be used. In this section, we discuss these issues with respect to GPFS.

2.4.1 Round-robin or segmented?

The two most prevalent ways of doing intra-file mapping are Round-Robin and Segmented data partitioning. Let us assume that we have an application with four tasks, and each task is performing a calculation that results in one GB of state that is to be committed to secondary storage, and that the application runs until the state is refined four times. The information written from each task is 4 GB (one GB four times), and the aggregate information for all four tasks is 16 GB. *Round-Robin partitioning* (also known as strided partitioning) would write the information in the following way: task0_firstGB, task1_firstGB, task2_firstGB, task3_firstGB, task0_secondGB, task1_second_GB, task3_fourthGB. On the other hand, a *Segmented*

partitioning would partition the file as follows: task0_firstGB, task0_secondGB, task0_thirdGB, task0_fourthGB, task1_firstGB, task1_secondGB, task3_fourthGB. The following figures illustrates both patterns.

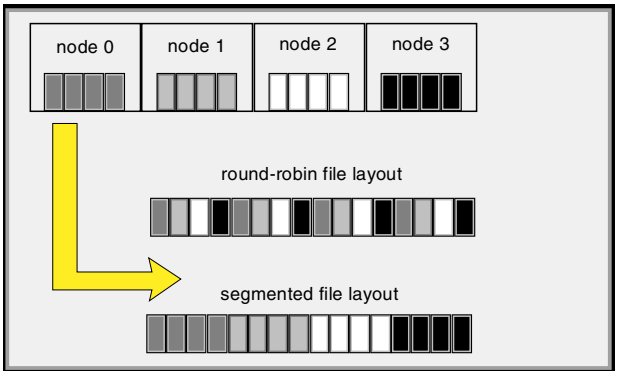


Figure 20. Segmented and round-robin file layouts

GPFS prefers segmented partitioning. This is because it results in sequential access for each task, therefore, permitting read-ahead (prefetch) strategies on reads and write-behind strategies on writing. For reads and writes much smaller than the GPFS blocksize (usually 256 KB), the performance difference can be significant especially for Round-Robin partitioning. For reads and writes much larger than the GPFS blocksize, the performance difference will be negligible (see also 2.5.2, “Local to global transformations” on page 66).

2.4.2 One file or multiple files?

A second data partitioning issue is how many files should be utilized. The primary GPFS factor involved here is the metadata updates to the directory block. When a new file is opened, GPFS updates the metadata in the directory block. Like other disk objects, a node must acquire a write token to update this metadata. Therefore, if the application has 1024 tasks that all wish to open a file in the same directory at the same time, a significant overhead for token acquisition will result.

If multiple files within the same directory are desired, one process should create all the files and then close them. Then, after one node has made all the changes to the directory block, other nodes can open the files for updates.

A second way to avoid the directory block contention problem is by using only one file. In this scheme, the application programmer must decide on an intra-file data partitioning scheme (see 2.4.1, “Round-robin or segmented?” on page 61).

Another factor that should be considered when choosing between one file and many files is convenience. If the files are to be post-processed, can the post-processing software work with a portion of the data? Will the user be required to keep track of thousands of files? Convenience issues may ultimately decide which partitioning scheme is used.

2.4.3 Using files larger than two gigabytes

Beginning in AIX Version 4.2, the operating system allows files that are larger than 2 gigabytes (2 GB). The AIX file system programming interfaces generally revolve around the off_t data type. In AIX Version 4.1, the off_t data type was defined as a signed 32-bit integer. As a result, the maximum file size that these interfaces would allow was 2 gigabytes minus 1. Beginning with AIX Version 4.2, the operating system provides two different ways for applications to be enabled for large-file access. Application programmers must decide which approach best suits their needs.

The first approach is to define _LARGE_FILES, which carefully redefines all of the relevant data types, structures, and subroutine names to their large-file enabled counterparts. Defining _LARGE_FILES has the advantage of maximizing application portability to other platforms since the application is still written to the normal POSIX and XPG interfaces. It has the disadvantage of creating some ambiguity in the code since the size of the various data items is not obvious from looking at the code. You can use sizeof(off_t) to determine the size of the datatype.

The second approach is to recode the application to call the large-file enabled subroutines explicitly. For instance, instead of using lseek(), one would use lseek64(). Recoding the application has the obvious disadvantages of requiring more effort and reducing application portability. It can be used when the redefinition effect of _LARGE_FILES would have a considerable negative impact on the program or when it is desirable to convert only a very small portion of the program.

It is very important to understand that, in either case, the application program MUST be carefully audited to ensure correct behavior in the new environment.

2.5 MPI-IO and GPFS

Message Passing Interface (MPI) libraries are a popular way of designing parallel applications. MPI is a standard for a message passing programming paradigm (contrast shared memory programming paradigm). As a standard, it should not be confused with any particular implementation. In fact, IBM has a supported MPI library designed and optimized for the SP series of machines.

Recently, the MPI standard was expanded at the request of the MPI user community. Among the new features of the revised MPI standard is a new chapter for parallel I/O commonly called MPI-IO. This version of the standard is called MPI2.

A full description of the capabilities of MPI-IO is beyond the scope of this section. What follows is a brief introduction to MPI-IO.

2.5.1 About MPI-IO

Any parallel I/O library requires several capabilities: (1) the capability to define the group of tasks that will take part in a collective operation. (In message passing parlance, an operation is said to be *collective* if more than one task participates in it and *independent* if only one task participates in it.) (2) synchronization primitives, such as barriers, (3) the capability to define derived datatypes, (4) the capability to transfer information between tasks on possibly remote nodes. People designing parallel I/O libraries immediately noticed that MPI already had all of this machinery. Furthermore, writing some bytes to a file is similar to sending a message, and reading some bytes from a file is similar to receiving a message. With this in mind, MPI was expanded to include MPI-IO in MPI2.

The basic unit within any MPI file is a *file type*. It is simply the template of where data will be placed for each read and write (file type should not be confused with the familiar concept of a document type, such as a text document or a binary document, and so on). The file type may consists of some number of atomic types (for example, floats or integers) or more complicated derived types built up from combinations of derived types. Each task has its own file type definition. A typical file type definition for a four-task application would define 1/4th the file type as in use and 3/4ths as void. This way, the file types of all four tasks, when merged, completely define all of the

bytes within the file type. The void portion of a file type is termed a *hole* in MPI-IO terms.

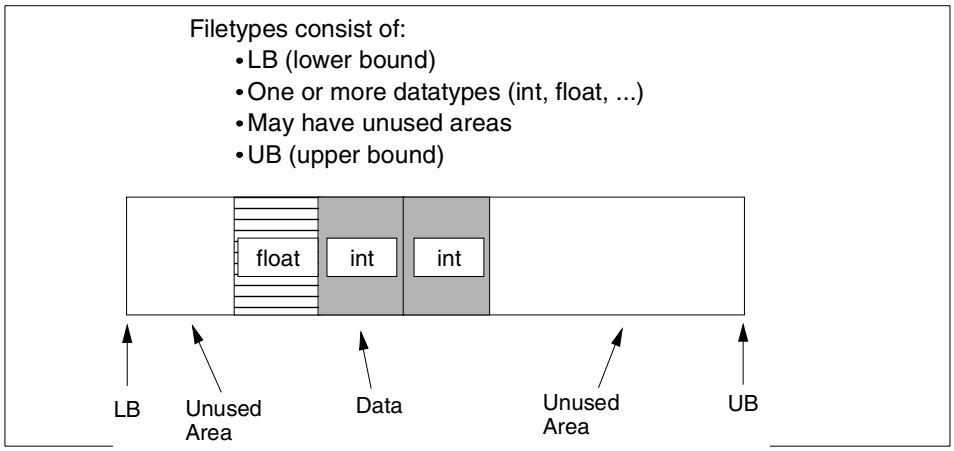


Figure 21. MPI-IO file types

An MPI file consists of an initial offset and then an infinite tiling of file types. Together, the initial offset and the file type define the file *view* for each task.

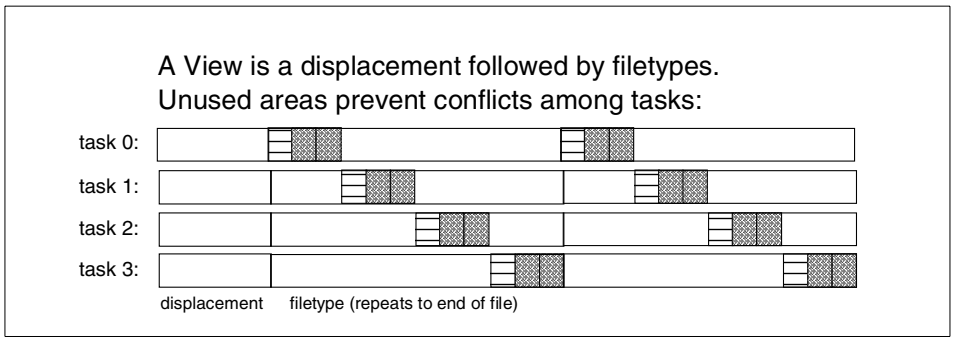


Figure 22. How an entire MPI-IO view is formed with file types

After a file is opened and a view is set, parallel I/O libraries can avoid much of the overhead of managing consistency conflicts. The view is able to specify that, although multiple nodes are writing to the same file, they never write to the same location.

2.5.2 Local to global transformations

File systems perform better with large reads and large writes. Many applications coalesce data to form structures that are contiguous and large. MPI-IO also has the ability to coalesce data and optimize writes. The question then becomes: At what layer should local to global transformations occur?

Unfortunately, there is no clear cut answer for this important question. Factors to consider are the maturity of the MPI-IO implementation, how efficiently coalescing can be performed at layers above MPI-IO, and how small the un-coalesced objects are.

As a rule of thumb, it is better to let lower layers perform as much work as possible because they have more knowledge of the actual workings of the file system. This assumes that adequate semantic knowledge of the problem can be propagated down to MPI-IO or lower libraries. That is, the view should define a contiguous item if all file types are present.

2.5.3 Application buffering

Many older applications that are I/O intensive have their own caching scheme implemented at the application level. However, since GPFS is also performing caching, any work done at the application level may actually be detrimental. For this reason, application or library caching above the GPFS level is deceptively tricky and should be avoided without a thorough analysis.

2.5.4 Hints support

MPI-IO defines a mechanism for specifying optimization parameters in a portable way. For instance, if the application is able to supply information to the file system that will enable it to be more effective with its prefetch algorithm, or when to release old blocks, better performance can result.

At present, GPFS ignores all hints. However, it is a good idea to provide hint information anyway since they will not hurt performance, and future versions of GPFS and IBM MPI may be able to make use of them.

2.6 On designing other I/O libraries with GPFS

Just as software libraries have been developed to export functionality in mathematics, message passing, and so on, a number of libraries have been developed for I/O. This section provides guidance for the I/O library developer.

2.6.1 Portability concerns

The application programming interface (API) to GPFS is POSIX; no GPFS-specific system calls are needed or provided. This means that applications using standard POSIX functions will run on GPFS without modification or recompilation. Several common non-POSIX functions are not provided with GPFS 1.2: mmap, munmap, and msync. In addition, since the atime/mtime/ctime information is maintained in a distributed manner (for performance reasons), some time is required before the most up-to-date information on an actively changing file is available to all nodes.

I/O libraries intended for use on parallel machines may wish to consider writing to the MPI-IO library (see Figure 23). As a parallel I/O standard, MPI-IO is expected to provide portability and high performance.

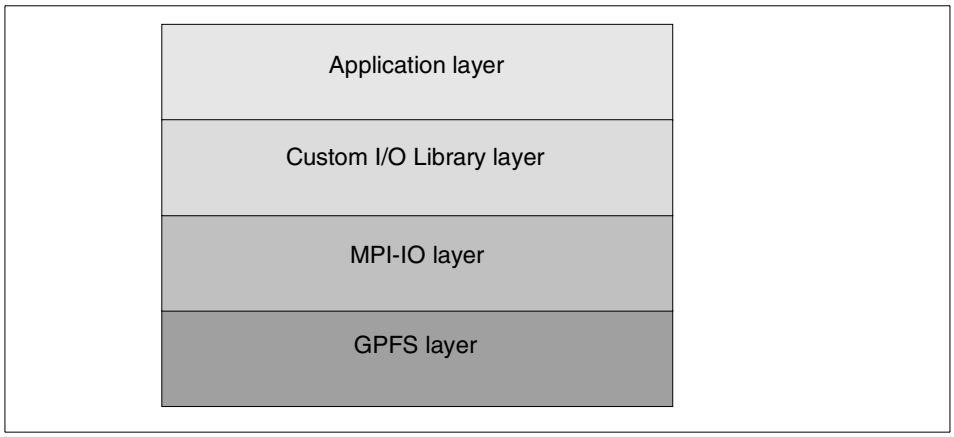


Figure 23. Diagram showing I/O library layers

2.6.2 Exposing GPFS internals

GPFS exports only one special ioctl system call for preallocation, which is documented in *General Parallel File System for AIX: Installation and Administration Guide*, SA22-7278. In this respect, it is different than PIOFS. All file system tunables (see Chapter 4, “Tuning GPFS” on page 101) are adjusted through system administrator utilities.

Likewise, GPFS does not export any non-POSIX internal data structures.

2.6.3 Threads, signals, and communication issues

GPFS is a thread-safe AIX threads implementation. That is, library developers do not need to be concerned with thread conflicts or scheduling problems when running on GPFS.

GPFS does not capture any additional signals. In particular, SIGUSR1 and SIGUSR2 are available for upper libraries. Signals are not recommended for message passing or shared memory programs and libraries.

Communications over the SP Switch adapter occur over one of six windows. One window is used for IP communications; one window is used by AIX for internal system communications; and four additional communication windows are available for applications using User Space (US) protocols, (see Figure 24).

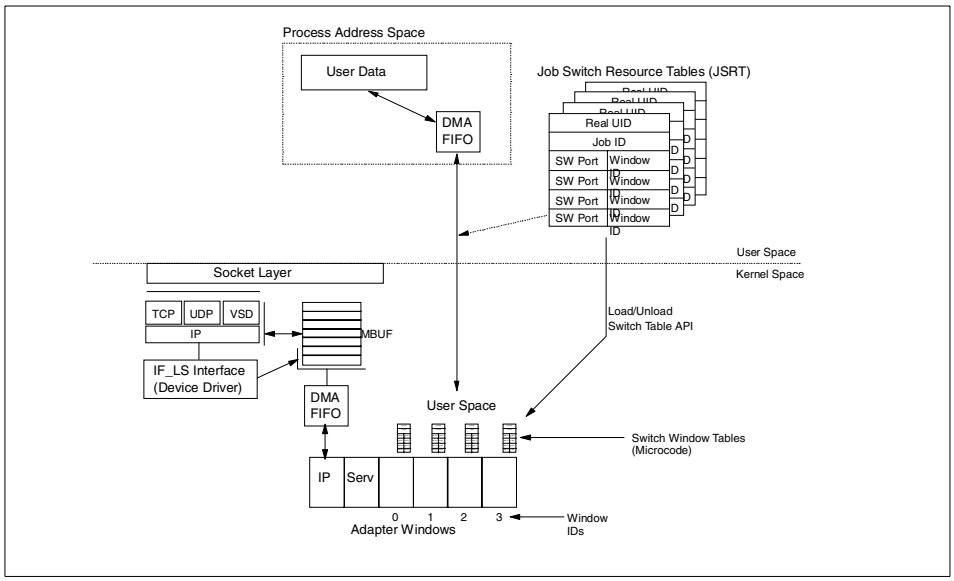


Figure 24. Adapter communication windows

Since IP permits multiple sessions over the same window, multiple applications can concurrently use the IP window. GPFS uses IP over the switch for data management and synchronization, which means that the IP window and all four the US adapter windows are available to applications when GPFS is used.

2.7 Analyzing an applications I/O

Several tools are available for monitoring and analyzing application I/O patterns.

2.7.1 AIX trace

AIX provides a tracing mechanism that is capable of providing extensive information on a specific application's I/O patterns. The utility is turned on a given node by a privileged user. Immediately, it begins capturing user specified information into memory.

2.7.2 Pablo

The Pablo research group at the University of Illinois at Urbana-Champaign has developed a variety of software tools for performance analysis and optimization of parallel and distributed systems. The resulting software distributions are intended primarily for academic and government research sites. Included in the tools are libraries that produce I/O traces and post-mortem tools that analyze the traces. You may freely retrieve, use, and modify the Pablo software as long as it is not for commercial gain. For more information, see:

`http://www-pablo.cs.uiuc.edu`

2.7.3 Monitoring file system activity

A number of tools exist for analyzing all activity on the file system. Care must be exercised when using this approach to assess a single application since the results of all applications running on the file system are reported. Still, this can be an effective technique. See 4.2.1, "Monitoring at the server" on page 135, and section 4.2.2, "Monitoring at the client" on page 144 for further details on this approach.

Chapter 3. Sizing GPFS

This chapter considers both the concepts and the recommended steps in planning and sizing a GPFS file system. This chapter approaches the subject from the point of view of someone who already has a specific performance requirement. Therefore, this chapter will guide the reader through the important decisions that have to be considered on the road to achieving a detailed GPFS configuration.

This chapter also focuses on these steps necessary to transform a known file system specification into a practical GPFS configuration.

3.1 Sizing concepts

Sizing, in concept, is very simple. Through sizing, you want to know what configuration will best fit your application requirements. Performance and efficiency are a given since most people would want to have the best performance at the lowest cost possible. But what about availability? Are you considering this when designing your GPFS configuration?

If availability and recoverability are issues in your application requirements, they have to be consider in the sizing exercise, and there is eventually a trade-off between performance, cost, availability, and recoverability.

There are two scenarios where sizing will most likely be executed. The first scenario is when sizing is done at the planning phase of an implementation. In this phase, there is some freedom to choose from different hardware configurations given a predetermine cost range and hardware limitations.

The second scenario is an existing configuration where GPFS is to be implemented. This scenario is quite common among RS/6000 SP installations since GPFS offers many benefits that existing customers would like to take advantage of.

Although both scenarios are perfectly valid, we will concentrate on the first one, when sizing is done at the planning phase, since this gives us more room and flexibility for theory and examples. If you are interested in the second scenario, you can always modify and adjust the assumptions made in this section to fit your own environment.

There are different approaches to the sizing problem if you consider sizing during the planning phase instead of sizing for an existing system. On an existing system, the most important question to answer is: How much can one

get from GPFS in my current configuration? This means that, given the amount of resources that you can allocate to GPFS, you would like to know how much data can be driven by GPFS. For this scenario, you may be restricted in terms of additional hardware and changes in the configuration itself; so, you would probably benefit more from a tuning methodology than from sizing concepts. However, sizing an existing configuration will give you the opportunity to check your current implementation against a configuration calculated in terms of your application's requirements.

If you are planning to implement GPFS for a given application, and you are wondering how much resources you will need in order to meet your application's requirements, then sizing should give you a good set of parameters so that you can balance between performance, cost, availability, and recoverability.

Any sizing exercise for a GPFS implementation should consider at least three important components:

- Data
- Metadata
- Disk servers

These three components of a GPFS implementation should be evaluated and configured considering the following factors:

- Performance
- Cost
- Availability
- Recoverability

The following sections provide in-depth discussions of the trade-off and considerations against these factors when sizing GPFS.

3.1.1 Data

Applications have different requirements about how data should be made available. GPFS and the underlying VSD technology and AIX provide several alternatives for data management. To select an alternative, you need to first address several issues.

The first requirement for data access you should consider is size. This is: How much data does my application have or produce? This will give you an initial pointer to other questions, such as: How many disk should I use? Or: How many servers do I need?

The data size will influence the decision for availability and recoverability as well as performance. Cost will probably be your limiting factor; so, let us consider space requirements first.

3.1.1.1 Space considerations for data storage

You should get a fairly good estimate in terms of data size from your application. Consider the amount of data required for your application and also the importance of this data. For example, determine if the data will be produced by the application, and the file system will be used as a dump area (stage area), or if the data will be downloaded to GPFS so that the application will just require access to it.

Will GPFS be a working file system? Or, is it going to be a repository for data?

The answer to these two questions will give you an idea about how much space your application will need. If GPFS will be a working file system, then you may have to double the amount of data required by the application assuming that temporary data will be placed in GPFS.

Assuming that, somehow, you have determined the amount of data you will put into GPFS, the next question is: Should I put that data in a single GPFS file system? Or, should I create several file systems?

Single file system versus multiple file systems

Although GPFS gives you the ability to create multiple file systems, you should not use the same logic as you do when creating local (JFS) file systems. Keeping multiple smaller file systems may seem to be a better choice when it comes to recovering a file system (think how much time would it take to run a file system check over a 5 TB file system); however, the rationale for GPFS is different than for other file systems.

A GPFS file system has a Stripe Group Manager (SGM) and a Token Manager (TM) assigned by the Configuration Manager (CM) when the file system is created. The file system is created over a set of VSD disks, which, instead, are served by VSD server nodes.

When an application on a node writes to a file within the file system, and assuming that the write will required additional data blocks to be appended to that file, the GPFS daemon running on that node will use the allocation maps (locally available) and communicate with the SGM to allocate the required blocks. In a single file system, this implies that the requirements over the SGM will be higher than having multiple file systems, thus, multiple SGM, to manage the same number of requirements.

Token management is another factor to consider. Since all requirements for access to a file within a file system are handled by the Token Manager; multiple requirements, even non-overlapping file requests, will impose a higher demand over the Token Manager compared to the situation where multiple file systems, meaning multiple Token Managers, could manage those requirements.

However, multiple file systems have a shortcoming that needs to be addressed too. Given that there probably is a fixed number of disks allocated to data (physical disks), and assuming that each physical disk contains a single VSD disk (GPFS will configure one VSD disk per physical disk. If more than one VSD disk per physical disk is required, it has to be done manually), multiple file systems imply less number of disks per file system, which translates into less raw bandwidth available for I/O operations.

So, the answer to single file system versus multiple file systems should consider all these factors. Recoverability implies that you should create smaller file systems, therefore, reducing the time needed to recover a single file system. Performance implications tell you to maximize the number of disks (total I/O bandwidth), which implies bigger file systems, thus, less numbers of them. For methodology and examples on sizing factors, refer to 3.2, “Sizing methodology” on page 85.

Storage options

Once you have determined the amount of disk that your application requires, you need to select the appropriated storage option. GPFS, along with VSD and AIX, offers several alternatives for storing data. These alternatives are:

- Plain (JBOD)
- Replication
- Mirroring
- RAID-5

Each one of these alternatives offer a trade-off between performance, cost, availability, and recoverability. The selection of one over the others will depend on performance and your cost limitations. Let us analyze the alternatives.

Plain (JBOD) — This alternative is the cheapest and simplest of all others. It offers the best performance and disk utilization. However, it does not offer data protection. If one of the disks fails, then some data will be lost with no possibility of recovery. Use this alternative when the data stored in GPFS can

be regenerated; so, GPFS is just a working area (or stage area). If your data is unique and critical, then this option is not recommended.

Replication — This alternative gives you some degree of recoverability since GPFS makes additional copies of your data and metadata and stores them in different disks and servers (known as *Failure Groups*). GPFS allows you to use replication at file level; so, there is not need to replicate an entire file system (doubling the disk space required) but only those files that contain critical data. Although the cost of implementing this alternative may be the lowest of all other alternatives that offer some degree of data protection, the performance penalties must be considered. Since replication is done at GPFS level, each write will be translated into two writes to the VSD servers. This implies that traffic coming out from the GPFS node towards the VSD servers will double, therefore, increasing the total bandwidth required on the node as well as on the SP Switch. If an application writes 20 MB/sec over a file that is replicated, then the I/O bandwidth requirements on that node will be double (40 MB/sec) since the GPFS daemon on that node will have to copy the data and metadata twice on different failures groups.

Mirroring — This is an excellent alternative if cost is not your limiting factor. Mirroring offers the easiest way to provide a good degree of recoverability at an extremely high cost. You will need to double the space requirements in order to implement this alternative. Since mirroring is done at the Logical Volume (LV) level, you need to mirror each VSD disk. Performance is also affected by mirroring. Although, mirroring is transparent to GPFS and VSD, the I/O requirements at the VSD server side are higher since one write on a particular VSD disk will involve two writes over the same I/O bus. Let us say that a VSD server receives a constant stream of data (10 MB/sec) for VSD disks that are mirrored. Assuming the best case, where each mirror copy is connected to a different adapter, then the I/O requirements will be 10 MB/sec in each adapter, which implies 20 MB/sec that the server has to handle for that particular stream of data.

RAID-5 — This is probably the alternative that offers the best mix between performance, recoverability, and cost. RAID-5 is done at the adapter level; so, the I/O requirements for a particular VSD server are not higher than the JBOD alternative. However, there are disk utilization and performance issues that need to be considered. Since RAID-5 is based on parity, not all the disk space will be available for data. Considering the popular 4+P (four data disks plus one parity disk), 20 percent of the disks have to be dedicated to store the parity. Performance in RAID-5 is worse than JBOD since the adapter has to calculate the parity before writing the data to disk. Besides, data block size is key to good RAID-5 performance. Since the adapter stripes the data across all data disks (four disks in case of 4+P), it uses a fix stripe size of 64 KB,

which implies that in a 4+P configuration, 256 KB block size for a write is the best choice. Not having the block size aligned with the full stripe size will cause the adapter to execute a *Read-Modify-Write* cycle, which means that two I/O operations are required to write the data to disk.

3.1.2 Metadata

Metadata is the name given to the disk blocks dedicated to store GPFS control structures, such as i-nodes and indirect blocks. Metadata activity will be largely influenced by the application's access pattern and the overall activity on the GPFS file system.

In terms of sizing, metadata does not play a very important role, but it is necessary to consider some aspects, such as placement, accessibility, and availability, where metadata may become a key factor to GPFS performance and recoverability.

3.1.2.1 Space considerations for metadata storage

Space requirements for metadata depend on some file system parameters and file system utilization. The parameters are as follows:

I-node size — The i-node size can vary from 512 bytes up to 4 KB.

Indirect block size — This number has to be a multiple of one subblock (BlockSize/32) and cannot exceed 32 KB or BlockSize, whichever is smaller.

Replication — Can be set for data and metadata. GPFS allows a maximum of two copies of data and metadata.

To calculate the amount of metadata that a given file system will require, you need to get some estimates in terms of the maximum number of files and the average file size.

The following are the metadata components:

- File system logs — These are 128 KB each and always replicated. This means that if metadata is replicated then there will be four copies of each log. Logs are per node that has the file system mounted. For example, if there are ten nodes that have the file system mounted, and replication is being used, then there will be 128x2x2x10 (5,1 MB of space used in log files).
- Disk allocation maps — These are always replicated. There 16 bytes allocated per block. For example, if you configure a file system of 200 GB of space, with a block size of 256 KB, then you have 819,200 blocks, which means that the disk allocation maps will use 25.6 MB.

- Inode allocation maps — These are always replicated and use 2 bits per i-node. For a 200 GB file system, you will probably configure 50,000 inodes, which means 25 KB of space.
- Inodes — These are replicated according to the file system parameter -m. For 50,000 i-nodes and a size of 512 bytes, the space required for i-nodes is 25 MB.
- Indirect blocks — These are replicated according to the file system parameter -m. The total amount of space used by indirect blocks will depend on the number of files and the average size. The space require for indirect blocks in a file will be the number of data blocks required multiplied by the size of each block pointer (6 bytes):

  $$\frac{6 \times FileSize}{BlockSize}$$

  The result must be rounded to the next multiple of the indirect block size.

  This formula is valid only for files whose size exceeds the i-node's capacity for direct block pointers. Each i-node has header information that occupies 104 bytes (it contains file size, ownership, dates, replication, permissions and several other administrative information), which leaves the rest of the i-node space for block pointers. For example, for an i-node size of 512 bytes, the remaining portion of the i-node is 408 bytes, which allows 68 block pointers. If the file is small enough to fit within 68 data blocks (17 MB for a BlockSize of 256 KB), then there will not be any indirect block used by the file. If the file does require more than 68 data blocks, then there will be a number of indirect blocks allocated to the file until the required space is satisfied.
- Directories — These are replicated according to the file system parameter -m. Directories are allocated in blocks of 8 KB. For example, for a file system with 512 directories, there will be 4 MB or 8 MB replicated used by directory blocks.

Let us assume there is an average file size of 20 MB and a maximum number of files of 10,000, which implies that the file system size has to be at least of 200 GB. Using Table 1 on page 12 from *GPFS for AIX: Installation and Administration Guide*, SA22-7278, we get the following values for the file system:

- In case of no replication
  - Block size = 16 KB

- Indirect Block Size = 4 KB
- I-node Size = 512 bytes
- Replication (M,R) = 1
- Maximum file size = 717.5 MB
- Maximum file system size = 478.4 GB

With these values, the amount of disk required for metadata is:

- File System logs — 128 KB x 2 x 10 (assuming 10 nodes mounting the file system). Total 2.56 MB
- Disk allocation maps — 819,200 blocks. Total 25,6 MB
- Inode allocation maps — Assuming 50,000 i-nodes. Total 25 KB.
- I-nodes — Assuming 50,000 inodes. Total 25 MB.
- Indirect blocks — Since the average file size is 20 MB, this means that there will be two indirect blocks allocated per file. Total 78 MB.
- Directories — Assuming 512 files (20 files per directory in average), it requires 4 MB of space.

The total amount of space for a non-replicated file system of 200 GB and the specifications previously given is 136 MB.

- In case of replication, the space needed is doubled to 272 MB.

As you can see from the calculations, the amount of disk required for metadata is less than 1 percent for a non-replicated file system and about 1.4 percent for a replicated file system. So, disk space requirements for metadata should not make any difference when it comes to size a GPFS configuration. However, metadata placement and accessibility may be an issue for systems with heavy metadata utilization, such as ADSM or file searching tools.

3.1.2.2 Performance considerations for metadata management

Metadata may have a strong impact on GPFS performance if the file systems have a heavy utilization, such as number of open files, directory listing, file searches, and file movements. Heavy metadata usage will impact performance. For example, if you list a directory (using the `ls` command) with thousand of files, GPFS will be extremely slow the first time compared to a local file system. This is because GPFS has to read many small blocks of data (i-node information) in order to provide the `ls` command with the required information for displaying the directory contents. The second time, GPFS will probably outperform the local file systems, and this is because the data may be already present in the mallocsize buffer on the client node.

Metadata management is done by the Metadata Manager (MM), which is the GPFS daemon running on the node that first opened or created the file. It will continue to be the MM until all nodes have closed the file.

Metadata activity consists in small blocks being read and written by the GPFS daemons. Each GPFS node will cache metadata, and each modification to a file will involve making the cache metadata invalid for that file. When multiples nodes open and work on the same file, there are two components that affect performance: Metadata activity and token management.

Since metadata activity consists of small blocks being transferred, scenarios, such as RAID-5 for metadata storage, need some additional configuration.

In previous sections, we stated that one of the best possible scenarios for GPFS, in terms of costs, performance and recoverability, is RAID-5. In that case, GPFS and applications should make their block size multiple of the full stride for optimum performance. For metadata, it is not possible to comply with this requirement; so, metadata management on RAID-5 has a penalty on performance. However, the performance impact needs to be analyze in several perspectives.

From a theoretical point of view, it seems to make sense to separate metadata from data when RAID-5 is being used with metadata intensive applications. This would allow to dedicate some raw disks for metadata exclusively, therefore, avoiding the inefficiency of transferring small blocks in and out of RAID-5 devices.

Although this make some sense, it is necessary to consider other factors that will influence the overall performance, and, in most cases, will demonstrate that separating metadata from data in a RAID-5 configuration may even impact performance negatively instead of improving it.

Let us make the following example. Assume that you need a file system of at least 200 GB. For this file system, considering 4.5 GB disks, you will need 45 disks to store 200 GB of data. But, because RAID-5 is implemented, then you need to add 12 additional disks (45 divided by 4) for a 4+P implementation. This gives you a total of 60 disks to implement 12 RAID-5 devices (12 multiplied by 5).

The metadata size for a 200 GB file would be around 10 MB (without replication). Refer to 3.1.2.1, “Space considerations for metadata storage” on page 76 for details. For this amount of metadata, you just need one disk. Although it is a waste of space, we could dedicate one 4.5 GB disk to store

metadata. Actually, with just one 4.5 GB disk, we could store all the metadata that GPFS could ever handle.

Now, let us analyze what happens when GPFS handles metadata requirements. In a multi-disk configuration, metadata requirements can be handled by different disks on different adapters and on different VSD servers. GPFS can issue multiple metadata requests, and all of them can go in parallel to the VSD servers and physical disks. The I/O bandwidth provided by the multiple data disks is by far bigger than the bandwidth provided by the single disk connected to the single adapter that is served by a single VSD server. Refer to Chapter 6, “Test results” on page 199 for test results on metadata management.

The only way to get a similar performance, in a case like the one previously described, is to add an equivalent number of disks and adapters to handle metadata. In conclusion, if you are not willing to dedicate the same number of disks, adapters, and servers to metadata as you do with data, then it is much better to have data and metadata intermix and striped across all your disks. You may not get the optimum RAID-5 performance for your metadata operations, but it will be far better than dedicating a reduced number of disks to metadata.

3.1.3 Servers

Servers in GPFS are VSD servers. Since GPFS uses VSD as the underlying technology for disk access, a VSD server is consider a node that contributes with physical disks to a GPFS file system.

Determining the number of VSD servers needed for a GPFS file system is something that heavily depends on the I/O bandwidth required by the applications running on top of GPFS and the number of disks and adapters required to provide that bandwidth. See Table 4 for disk performance and Table 5 on page 81 for adapter performance numbers.

Table 4. Disk performance

Name	Capacity GB	Sustained Transfer Rate MB/s	Random 256 KB JBOD Transfer Rate MB/s	Estimated GPFS Throughput for JBOD MB/s	Estimated GPFS Throughput for full strided RAID-5 MB/s	Estimate GPFS Throughput for non-strided RAID disks MB/s
Starfire	1.1, 2.2, 4	7.2	5.46	3.82	2.21	1.11
Scorfire	2.2, 4.5	10.1	6.97	4.88	2.53	1.27
Scorpion	9.1	10.1	6.97	4.88	2.53	1.27

Name	Capacity GB	Sustained Transfer Rate MB/s	Random 256 KB JBOD Transfer Rate MB/s	Estimated GPFS Throughput for JBOD MB/s	Estimated GPFS Throughput for full strided RAID-5 MB/s	Estimate GPFS Throughput for non-strided RAID disks MB/s
Sailfire	4.5	15.4	9.15	6.40	2.89	1.44
Sailion	9.1	15.4	9.15	6.40	2.89	1.44
Sailfin Jr.	4.5	15.4	9.15	6.40	2.89	1.44
Sailfin	9.1	15.4	9.15	6.40	2.89	1.44
Marlin	18.2	15.4	9.15	6.40	2.89	1.44
Swordfish	36.4	20	10.59	7.42	3.08	1.54

For example, if a 200 GB file system required 57 4.5 GB disks for RAID-5 (as the example in 3.1.2.2, “Performance considerations for metadata management” on page 78), and we consider that 12 RAID-5 devices can drive up to 14 MB/sec (assuming 3.5 MB/sec per disk in 4+P RAID-5), we require 168 MB/sec if we want to fully utilize disk I/O bandwidth. This bandwidth clearly cannot be provided by a single server much less a single adapter. If we consider SSA technology (which is strongly recommended for GPFS implementations), we can divide the total bandwidth required by the bandwidth of a single adapter.

The capacity of a single SSA adapter will vary depending on factors, such as type, bus technology, and memory options. However, for sizing, we can rely on some rough numbers available for that purpose.

Table 5. Performance of SSA adapters

Adapter Feature Code	**Bus Type**	**Adapter Name**	**Max. Throughput JBOD MB/sec**		**Max. Full Stride Throughput RAID-5 MB/sec**		**Max. Non-Full Stride Throughput RAID-5 MB/sec**	
			WRITE	**READ**	**WRITE**	**READ**	**WRITE**	**READ**
6215	PCI	IBM Enhanced RAID	31	42	21	30	7	30
6225		IBM Advanced Serial RAID	85	90	75	90	21	86

Adapter Feature Code	Bus Type	Adapter Name	Max. Throughput JBOD MB/sec		Max. Full Stride Throughput RAID-5 MB/sec		Max. Non-Full Stride Throughput RAID-5 MB/sec	
			WRITE	READ	WRITE	READ	WRITE	READ
6216	MCA	IBM Enhanced	35	35	N/A	N/A	N/A	N/A
6219		IBM Enhanced RAID	35	35	22	32	7	33

A 6215 PCI adapter is capable of driving a maximum of 21 MB/sec in RAID-5 configurations (considering write case), but GPFS will see only a fraction of this. A safe estimate is considering 70 percent of the maximum throughput, which gives us 14.7 MB/sec per adapter. This means that for a 168 MB/sec requirement, we need at least twelve adapters (total 176.4 MB/sec). Obviously, we cannot put all the adapters in a single server; so, we need to balance the load across multiple adapters on multiple servers. Refer to Table 6 for node performance on GPFS.

Table 6. SP Switch and GPFS throughput

Node Type	Processor Type	Procs per node	Bus Type	TCP/IP max bandwidth Uni-di MB/sec	GPFS Throughput MB/sec
120MHz Thin	POWER2SC	1	MCA	87	60.9 (*)
135MHz Wide	POWER2SC	1	MCA	86	60.2 (*)
160MHz Thin	POWER2SC	1	MCA	104	72.8 (*)
112MHz SMP High	PowerPC 604e	2, 4, 6 or 8	PCI	33	23.1
200MHz SMP High	PowerPC 604e	2, 4, 6 or 8	PCI	46	32.2
332MHz SMP Thin	PowerPC 604e	2 or 4	PCI	109	76.3
332MHz SMP Wide	PowerPC 604e	2 or 4	PCI	109	76.3
200MHz P3 SMP Thin	POWER3	1 or 2	PCI	135	94.5
200MHz P3 SMP Wide	POWER3	1 or 2	PCI	135	94.5

(*) = 50 MB/sec effective due to MCA limitations.

Another alternative is to use the 6225 adapters that provide higher throughput. From Table 5 on page 81, we see that a 6225 SSA adapter can provide a maximum of 75 MB/sec for a full stride. However, the estimate GPFS throughput is a fraction of this maximum. A safe estimate is 70 percent of the maximum throughput, which gives us 52.5 MB/sec. This means that we

would only need three adapters (168 MB/sec divided by 52.5 MB/sec) to get the required bandwidth.

The number of servers required for the three adapters will depend on the type of nodes you choose. If you decide on 332 MHz SMP nodes, then you would need three servers (see Table 6 on page 82 for GPFS throughput) since these nodes will not give you enough throughput for more than one 6225 SSA adapter. If you decide to use POWER3 SMP nodes, then you may be able to put two adapters in one server and the third one in a second server, thus, reducing the number of servers required to two. However, three servers and three adapters seems to be a more balanced solution.

This rough calculation gives you an idea about the bandwidth required and provided by your configuration. It does not mean that you have to go ahead and install the required number of VSD servers and adapters. What it means is that given your initial requirements of a 200 GB file system, the number of disks and adapters required for maximum performance is the one calculated. But, there are other factors that you need to consider before you decide on a particular configuration.

3.1.3.1 Client considerations

Calculating the number of VSD servers required for a specific configuration only gives you half of the picture: The server's half. GPFS is not a client/server type of file system; however, the relationship between GPFS and VSD is client/server. Nodes running GPFS behave as VSD clients when they access data and metadata.

By calculating the number of VSD servers, you have determined the maximum bandwidth that your disks (from GPFS viewpoint) can give you. However, the number and quality of clients will determine how much data you can get out of those disks.

There are four factors that are going to limit client throughput. These are:

- CPU
- Pagepool buffer
- Worker threads
- SP Switch throughput

CPU — The GPFS daemon (mmfsd) and the GPFS kernel extension will consume a certain percentage of CPU per MB/sec transferred. This percentage of CPU needs to be evaluated in order to get an estimate of how much CPU would be left for the application to produce the data to be transferred. For example, if we assume that 1 percent of CPU is used by

GPFS per MB/sec transferred, then to get the maximum throughput per node (considering POWER3 SMP nodes from Table 6 on page 82), GPFS would utilize almost all CPU for data management, which means that the application will have little time to produce it. Refer to 6.5.7, “Analysis of CPU usage with regard to dedicated VSD servers” on page 216 for estimated numbers of CPU utilization.

Pagepool buffer — The amount of memory available to GPFS for write-behind and read-ahead algorithms will affect the overall performance of GPFS. Reducing the pagepool size will reduce GPFS throughput, which will lower the I/O bandwidth required from the VSD servers.

Worker threads — These are threads from the mmfsd daemon allocated to do data and metadata management. The number of worker threads can be increased or decrease to adjust the overall throughput per node. This may be a good technique to throttle down clients when VSD servers are being overrun.

SP Switch throughput — When VSD servers are located in nodes other than GPFS nodes, the SP Switch network is used to transfer data to and from the server. The maximum SP Switch throughput per node, listed in Table 6 on page 82, will limit the maximum amount of data that a client can drive in and out of VSD servers located remotely.

Considering the factors previously mentioned, you can estimate the number of clients required to drive all the data throughput provided by the VSD servers. It could also allow you to estimate the I/O bandwidth required on the server side, given a configuration on the client side.

3.1.3.2 Server performance considerations

There are at least two factors that limit the throughput of a VSD server. The I/O subsystem, which limits the capacity of the server to deliver or receive data, and the CPU, which limits the capacity of the server to process data and control.

From the test results in A.1, “Base runs” on page 225, we can see that the CPU percentage to MB/sec ratio is about 0.6 to 1 on a VSD server and 1 to 1 on a GPFS client. These number are valid for non-replicated file systems with a sequential access pattern.

For more information, refer to 6.5.7, “Analysis of CPU usage with regard to dedicated VSD servers” on page 216.

3.2 Sizing methodology

This section will cover the recommended steps for transforming an initial file system specification into a practical GPFS configuration. This methodology is aimed at those users who require good performance from a single large file system rather than those users who are using GPFS mainly for its ability to be mounted on any node and, therefore, may be more interested in having multiple, smaller file systems.

It is important to size your GPFS system with an adequate throughput to cope with all of its clients requirements. If the GPFS file system is overloaded by the total of the GPFS client node demands, it can become unavailable, and the accessing GPFS client node applications may fail.

Sizing then becomes mainly concerned with ensuring that the capacity of the VSD server nodes for data throughput is adequate to meet the demands of all the GPFS client nodes. An aspect of sizing that is not covered here is the sizing of the capacity of an individual GPFS client node given a known VSD Server capacity.

Here we will not consider the metadata separately, as we will assume that the metadata is always mixed in with the data, and this will spread the metadata on the most disks and make administration of the file system much easier. The space taken by the metadata for large files is so small in comparison with the data space that its effect on the total file system capacity is negligible.

We will also assume here that each VSD is on a single disk and no two VSDs share a disk. This is the default action if you let the `mmcrfs` command create the GPFS file system. It is possible to create multiple VSDs per disk as may be desired if constructing many smaller file systems on a few disks. This, however, makes the job of predicting performance almost impossible due to the contention for I/O resource by the VSDs on the one disk caused, perhaps by activity on different file systems, this. Therefore, this is not considered here.

Another assumption made here, for the purposes of sizing, is that all GPFS client node applications are using sequential I/O access and are not random.

In this section, we provide tables whose aim is to help with estimating practical performance and, hence, sizing issues. The tables incorporate a common theme. That is, they try and identify, for a key component that GPFS relies upon, the maximum bandwidth that components can sustain. The GPFS throughput rate, for this component, is then shown to be 70 percent of the maximum. This rule of thumb reflects the fact that because of the nature

of the job that GPFS is doing, it is unable to use any particular component to its optimum extent and also allows some margin of error in assuring that a sized GPFS system can accommodate the performance required.

Here are the basic steps involved in sizing:

1. Find out the requirements of the file system
2. Recoverability considerations.
3. Determine minimum number and type of disks to give the required performance.
4. Check and adjust for File System Capacity.
5. Determine the number of VSD servers required.
6. Determine the detail of the VSD server configuration.

First, we will cover the theory of each step. Following that, we will illustrate the steps using worked examples.

3.2.1 Step 1 - Find file system requirements

This is the most important step. You need to be clear about the requirements for data throughput that you are trying to satisfy.

Find the following for your file system:

- File system capacity
- Number of clients requiring access to the file system.
- The workload requirements of each client. This is in terms of the following:
  - Total Read performance in MB/sec
  - Application's request block size for reads - (large block > 64 KB for best performance)
  - Type of read access - (full block sequential is best)
  - Total Write performance in MB/sec
  - Application's request block size for writes - (large block > 64 KB for best performance)
  - Type of write access - (sequential is best)

In order to achieve the best performance that a GPFS file system can give, you have to use sequential access using large block size. Using either random access patterns or block sizes less than 32 KB, you will be limited to a fraction of the performance that is possible.

When formulating the individual client requirements, be careful not to over estimate the throughput available to a client. From measurements taken in the results section, it would seem that the maximum throughput for a single client is about 70 percent of the estimated GPFS Switch throughput for that node type quoted in tableTable 6 on page 82.

Having collected this data, use these figures to work out the maximum aggregate bandwidth seen by the file system for all the clients. Do this for both the writes and the reads. The highest of the two we will call the *max file system bandwidth*.

This figure will have to be increased if the block size being used for either reads or writes is other than 256 KB. The factor to use can be approximated using the GPFS performance figures in Figure 25, shown below, or Figure 26 on page 88.

For example, if your application uses 32 KB read blocks, then increase the *max file system bandwidth* figure by a factor of 1/0.9 = 1.11

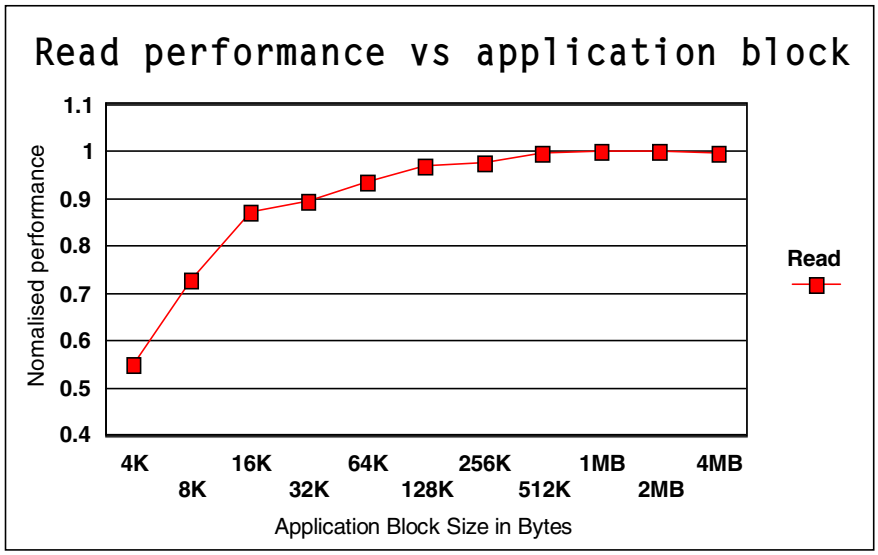


Figure 25. Performance of GPFS reads with block size

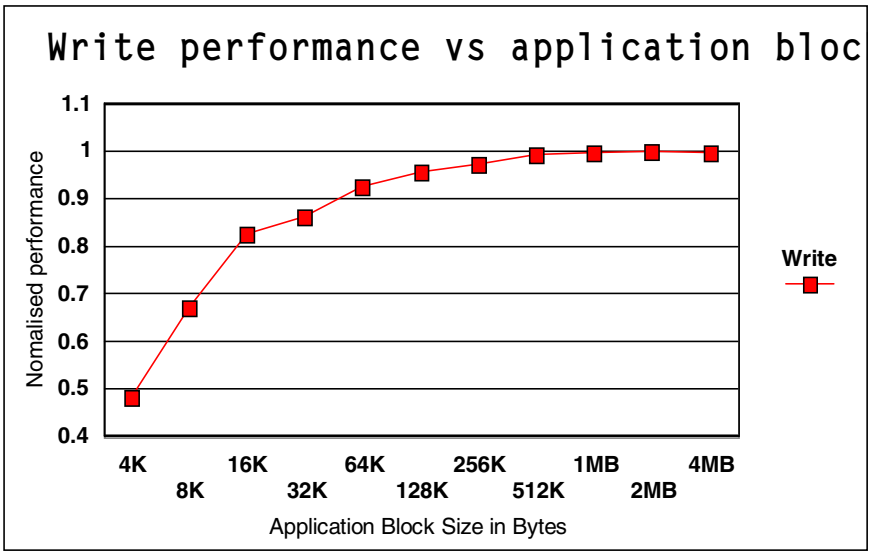


Figure 26. Performance of GPFS writes with block size

3.2.2 Step 2 - Recoverability considerations

It is important to decide on the type of file system. The options are

- JBOD
- RAID-5
- Mirroring
- Replication

The advantages and disadvantages are discussed in “Data” on page 72.

If you decide on either a Mirroring or Replication file system, then you have to adjust the figure for *max file system bandwidth* and the figure used for *File system capacity*.

In the case of Replication, the *max file system bandwidth* figure is simply doubled for use in the subsequent steps. This is because each read or write required by the GPFS client node will be doubled from the client node. The figure for *File system capacity* must also be doubled to allow for the extra disks space required.

In the case of Mirroring the *max file system bandwidth*, the figure is doubled for use in the all subsequent steps apart from step 5, where we are choosing the number of servers. This is because for each read or write required by the GPFS client node, a single request will be sent over the switch, but double the requests will go through the SSA adapters to the disks. The figure for *File system capacity* must be doubled to allow for the extra disks space required.

3.2.3 Step 3 - Determine minimum number and type of disks

To determine minimum number and type of disks, we must do the following:

- Decide on the disk type.
- Estimate the throughput rate for the chosen disk when running with GPFS.

To calculate this for any disk, you need to know the sustained throughput rate in MB/sec for sequential 256 KB block access-*R*, the seek time in milliseconds-*s*, and the rotational speed-*rpm* of the disk. First, calculate the throughput rate for randomized 256 KB block access, which is what GPFS will effectively use at the server, even for sequential client access. This is:

0.256/(256/(1000**R*)+(*s*/1000)+(30/*rpm*) MB/sec

where:

s/1000 is seek time in seconds.

30/rpm is average latency in seconds.

256/1000*R is transfer time of a 256 KB block.

This is the fastest access, in theory, but because GPFS has to allow multi-client access and must also deal with metadata and token management we have to allow for the fact that GPFS does not use every single disk to its optimum extent all the time. We, therefore, use 70 percent of this calculated figure as the maximum expected rate that GPFS can achieve with a single disk. For SSA disks, the data and the calculations have been set out for your convenience in Table 4 on page 80. Note that for 4+P RAID-5 arrays, the performance per data disk (not parity) is given separately for these and are quoted in the column entitled Estimated GPFS Throughput for single RAID-5 data disks - MB/s. This is calculated using the following formula:

(0.256/(64/(1000*B4)+(C4/1000)+(30/D4)))*.7/4

Note that here we again applied the 70 percent utilization GPFS factor for this calculation.

To help you identify SSA, “Disk performance” on page 80 is provided.

Table 7. SSA disk identification

Name	FRU Number	Capacity /GB	Type/Model	Device Specific (Z2)
Starfire	88G6195, 88G6196, 88G6198	1.1, 2.2, 4.5	DFHCCxB1 x is capacity	RAMST077
Scorfire	88G6197, 88G6199	2.2, 4.5	DFHCCxB1	RAMSC081
Scorpion	88G6200	9.1	DCHC09B1	RAMSC095
Sailfire	09L2273	4.5	DFHCC4x1 x is B or C	CUSMA903
Sailion	09L2274	9.1	DCHC09x1 x is B or C	CUSMA903
Sailfin Jr.	09L4294	4.5	DGHC04B	CUSMA903
Sailfin	09L4295	9.1	DGHC09B	CUSMA903
Marlin	09L4296	18.2	DGHC18B	CUSMA903

So, using the SSA Disk Throughput for GPFS in Table 4 on page 80 or a table you have constructed for your own disks, pick the throughput rate for the disks you are going to use. Unless you have chosen a RAID-5 file system, you must use the JBOD figures shown in the table.

Using this figure for throughput rate, we can simply divide the *max file system bandwidth* figure generated from the last step by the throughput rate and calculate the *minimum number of disks* that are required for performance.

3.2.4 Step 4 - Check and adjust for file system capacity

Having determined the *minimum number of disks* for performance, now divide the figure for *File system capacity* by the *minimum number of disks*. The actual size of the disk will have to be equal or greater than this figure. If they do not make disks that big, then you will have to further increase the total number of disks for the file system, which will improve performance.

3.2.5 Step 5 - Determine number of VSD servers required

This step can be further subdivided into the following steps:

- Choosing the number and type of SSA adapter cards

- Choosing the number and type of Servers

Choosing the number and type of SSA adapter cards

For particular SSA adapter, there is a maximum bandwidth that the adapter card will support. From Table 5 on page 81, you can see that for the 6215 SSA adapter, for example, the maximum read bandwidth is 42 MB/sec.

Table 8. GPFS throughput performance of SSA adapters

Feat Code	Bus Type	Adapter Name	Max. Estimated GPFS Throughput JBOD - MB/sec		Max. Full Stride Estimated GPFS Throughput RAID-5 - MB/sec		Max. Non-Full Stride Estimated GPFS Throughput RAID-5 - MB/sec	
			WRITE	READ	WRITE	READ	WRITE	READ
6215	PCI	IBM Enhanced RAID	21.7	29.4	14.7	21	4.9	21
6225		IBM Advanced Serial RAID	59.5	63	52.5	63	14.7	60.2
6216	MCA	IBM Enhanced	24.5	24.5	N/A	N/A	N/A	N/A
6219		IBM Enhanced RAID	24.5	24.5	15.4	22.4	4.9	23.1

Table 5, “Performance of SSA adapters” on page 81, reflects again a 30 percent reduction of the maximum rates and gives the different rates for use if using both RAID-5 and JBOD disks. It also shows the how much slower doing a non - full strided write to RAID is. To achieve a full stride write with RAID 5, the RAID arrays must be configured for 4+P, the file system block size must be 256 KB as well as the application write block size.

To be able to choose an adapter, you should have some idea of the node type that you will want to use. This is because a specific type of SSA adapter card will be compatible with only one bus type, PCI or MCA. Using Table 5, “Performance of SSA adapters” on page 81, pick an adapter of the appropriate bus type and note the *adapter throughput rate* that applies to your particular file system. For example, RAID-5 or Non RAID (JBOD). Note that this involves picking an adapter and bus type.

Divide the *max file system bandwidth* by the *adapter throughput rate* to determine the number of adapters you require.

The figure for number of adapters is only correct if you can equally divide the total number of disks by the number of adapter cards. For RAID-5, especially if you are using 4+P RAID-5 groups, the constraints imposed by the RAID-5 groupings may require a further increase in the number of disks.

Choosing the number and type of Servers

The nominal, uni-directional bandwidth of the currently available SP Switch is stated as 150 MB/sec. This bandwidth does not take into account the transmission protocol. The TCP/IP maximum uni-directional bandwidth in MB/sec at which data can be shipped from one node to another is shown in Table 6 on page 82. This throughput is not achievable by GPFS due to the way in which it operates. The same table also shows that the expected GPFS data throughput using the switch is 70 percent of the maximum figure. The highest GPFS throughput figure achieved for any node type is 94.5 MB/sec.

Having chosen the number of adapters required, use Table 6 on page 82 to pick a VSD node server type that is compatible with your adapter card. Now check how many adapter cards can be used in this node type. This information for PCI bus nodes can be checked using the RS/6000 SP PCI Adapter Placement Information Version 1.1. available on the Web at:
`http://cs2.austin.ibm.com/ibmsm/ibmsm.nsf/mainframeset?readform`

If you intend to use more than one adapter card per node, check, using Table 6 on page 82, that the aggregate GPFS data throughput available from both adapter cards is less than the GPFS throughput available over the switch.

Check that there is both sufficient aggregate switch throughput available and that the node type you have chosen can accommodate the number of adapters you need.

The Read GPFS throughput rate for the SSA adapter feature code 6215 is shown to be 29.4 MB/sec. This, therefore, means that we can use a maximum of two of these adapter cards per node before we hit the throughput limit imposed by the switch. This is the case for 332 MHz SMP wide nodes where they have two separate PCI buses, but it is not possible for a 332 MHz SMP thin nodes. Although we could physically install two SSA adapter cards in a 332MHz SMP thin node, the rules given in the RS/6000 SP PCI Adapter Placement Information Version 1.1. would prevent it as the bandwidth of each adapter would be constrained to less than its maximum due to I/O contention on the single PCI bus. The 332 MHz SMP wide node, having two buses, does not suffer this same problem and is allowed to have a maximum of two 6215 adapters.

If you are using RAID-5, you may be able to use two adapter cards per PCI bus, but, in general, you are limited to one per bus. Some nodes have more than 1 bus; so, if you choose these nodes, you may be able to have 2 SSA adapters in one node.

You should now know the total number of SSA adapter cards you need, the type of node to use, along with the number of adapter cards per node. Dividing the total number of adapter cards by the number of adapter cards per node will give you the number of Server nodes required.

3.2.6 Step 6 - Determine VSD server configuration

We now have sizings of the basic building blocks of a GPFS system, but there are still a few more sizing issues to be considered before being able to perform the initial configuration, fire up the file system, and hand it over to the tuning team. These issues are:

- Correct placement of adapter and disks.
- Ensure that the SSA disks are positioned to cause the least contention for SSA loop bandwidth.
- Set initial configuration parameters for the IP Network, the switch, SSA control, VSD servers, and GPFS nodes.

There are strict guidelines to following placement of SSA adapters. The rules found in the following Web site must be followed:
`http://cs2.austin.ibm.com/ibmsm/ibmsm.nsf/mainframeset?readform`

Disks should be shared out evenly amongst all the available SSA loops. Within a loop, the disks that are being served by a node must be on the shortest loop path to that node and not have their path obstructed by a disk on the loop that is being served by another node and could, therefore, be competing for loop bandwidth.

For the most part, the default parameters will suffice and are unrelated to size issues. There are, however, a few parameters that are size related.

First, we will list in the Table 9 the default parameters that are unrelated to size.

Table 9. Table of default sizing parameters to use for GPFS

Parameter	Setting	Comments
thewall	65536	Network
ipqmaxlen	512	Network
rpoolsize	16777216	Switch
rpoolsize	16777216	Switch
max_coalesce	256	SSA

Parameter	Setting	Comments
init_cache_buffer_count	64	VSD
max_cache_buffer_count	256	VSD
vsd_request_count	256	VSD
rw_request_count	48	VSD. Also referred to a pbuf
min_buddy_buffer_size	4096	VSD
max_buddy_buffers	1-GPFS Clients	VSD One required for all GPFS Nodes
vsd_max_ip_msg_size	61440	VSD
pagepool	40M	GPFS - default
mallocsize	4M	GPFS - default

Now, we will list in Table 10 the parameters that are size dependent.

Table 10. Table of size dependent initial parameters to use for GPFS

Parameter	Dependency	Comments
max_buddy_buffer_size	Max GPFS File system block size	This is set to the same as the maximum GPFS block size used to build the file system
max_buddy_buffers	VSD disks per Server	For each VSD Server node this should be 4 times the number of VSDs served by that node.

3.2.7 Sizing example 1

Step 1. Find file system requirements

These are the specification for our worked example:

- 140 GB file system
- Single client access with 20 MB/sec sustained write and 50 MB/sec sustained read performance. This is due to the nature of our application that has been written with GPFS performance in mind and uses 256 KB blocks to both read and write.
- Max number of concurrent clients is five clients
- Total bandwidth requirement is, therefore, 100 MB/sec write and 250 MB/sec read. Let us say that this cannot occur together, but we have to cope with maximum capacity of both read and write

This gives us a figure for max file system bandwidth is 250 MB/sec READ.

Step 2. Recoverability Considerations

The first thing is to decide if we are going to use RAID-5 for this file system. We will say no, as we keep regular backups of the data and would take no great hit if we lost all the data on this file system occasionally. We would rather get great read performance for the smallest cost.

Step 3. Determine minimum number and type of disks

We know that we can either buy some new disks, say 9.1GB disks, or we could use some old ones we already have that are 4.5 BG. Table 4 on page 80 shows that for the Sailfin 9.1 GB disks, we can use a GPFS disk throughput rate of 6.4 MB/sec. The calculation for Sailfin shows that we, therefore, would need 250/6.4 = 39 disks.

From looking at the identification on our old disks, we know that they are Scorefires, and for these we can use a GPFS disk throughput rate of 4.9 MB/sec. The calculation for Scorefire shows that we, therefore, would need 250/4.9 = 51 disks.

Step 4. Check and adjust for file system capacity

Using 39 Sailfins would give us 39x9.1GB=819GB of file space, which is much larger than our requirement.

Using Scorefires would give us 51x4.5GB=229.5GB of file space, again larger than our requirement but perhaps not so costly. We will choose this option.

Step 5. Determine number of VSD servers required

From Table 5 “Performance of SSA adapters” on page 81, we can see that the GPFS Throughput rate for the 6225 adapter is 63 MB/sec. We would, therefore, require about 250/63 = 4 of these 6225 adapter cards. If we chose the 6215 adapter cards, we would require 250/29.4 = 8.5, or 9 cards. We will go with the four 6225 adapter cards.

Now let us look at the server options. We could fit two 6225 cards per server so that we could just use two VSD Servers nodes. This option would give us a required single server GPFS throughput on the switch of 250/2 = 12 5MB/sec. Looking at Table 6 on page 82, there is no server listed that can support this throughput rate on the switch. The nearest we can get is 76.3 MB/sec using a 33 2MHz SMP wide node. So, it looks as if we will have to look to use four VSD Server nodes each with one 6225 card. These nodes would have to be able to provide a GPFS switch throughput rate of 250/4 = 62.5 MB/sec. From Table 6 on page 82, we can see that there are only three such nodes listed. The 160 MHz thin node is a micro channel (MCA) node and will not, therefore, accept the 6225 card.

As we only need one card per node, we can use a thin node. We choose the 332 MHz SMP thin node.

Step 6. Determine VSD server configuration

We now have decided on four 332 MHz SMP thin nodes each with one 6225 adapter card each. Using the 51 Scorefire disks we have would give us 12.5 disks per node. To make it symmetrical we could use 13 disks per node giving us a total of 56, but, unfortunately, SSA demands that the number of SSA disks on a loop is divisible by four. We could, therefore, have 14 disks per node using both loops of the Santa Cruz adapter card with one loop having 12 disk connecting two Santa Cruz cards on a pair of nodes and 16 disks on the other loop connecting the nodes (see “Example 1 SSA loop disk layout for a pair of connected nodes” on page 96). This gives a total of 28 disk connected between a pair of nodes. This configuration allows the use of the recoverability feature of the RVSD software, such that if one node of the pair fails, the remaining node can be set up to take over serving the VSDs owned by the failed node albeit at a reduced data throughput rate as both the single nodes switch and single SSA adapter throughput would considerably constrain the performance.

For such a configurations where SSA loops are connected between node A and Node B, it is very important to ensure that the disks used by each node have the shortest physical path to the owning node.

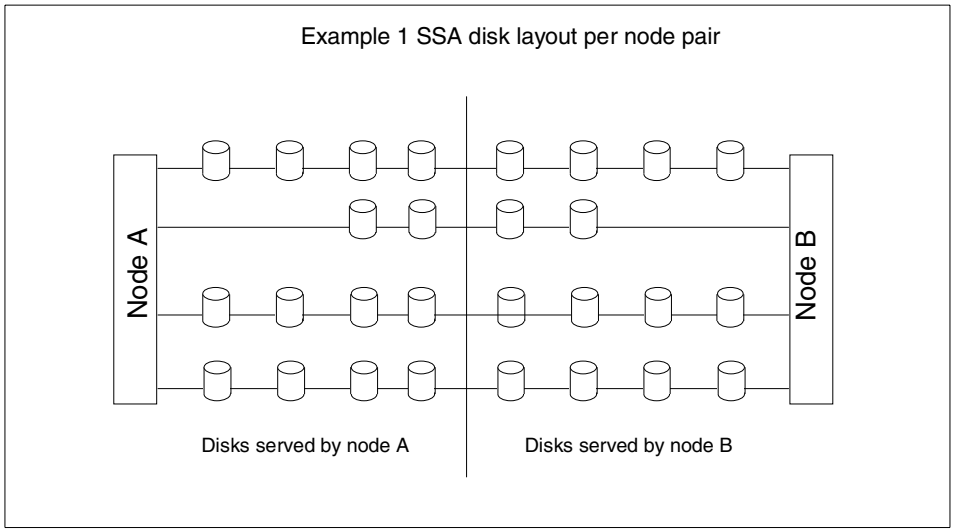


Figure 27. Example 1 SSA loop disk layout for a pair of connected nodes

3.2.8 Sizing example 2

Step 1. Find file system requirements

These are the specifications for our worked example

- 1.2 TB file system
- Max number of concurrent clients is 50 clients
- Read performance required per client - 10.8 MB/sec.
- Read block size - 16 KB
- Type of read access - sequential
- Write performance required per client - 8.7 MB/sec.
- Write block size - 16 KB
- Type of write access - sequential

Total bandwidth requirement is, therefore, 436 MB/sec write and 543 MB/sec read. Let us say that this cannot occur together, but we have to cope with maximum capacity of both read and write

Due to the read and write block size being less than 256 KB, we must increase this *max file system bandwidth* by a given factor different for read and write.

Using Figure 25 “Performance of GPFS reads with block size” on page 87 for reads, we use the factor 1/0.88 to get a read max file system bandwidth of 617 MB/sec. Using Figure 26 “Performance of GPFS writes with block size” on page 88 for writes, we use the factor 1/0.82 to get a read max file system bandwidth of 532 MB/sec.

The figure for *max file system bandwidth* that we must use for the sizing is now 617 MB/sec read.

Step 2. Recoverability considerations

The first thing is to decide if we are going to use RAID-5 for this file system. We will say yes, as the application runs are so long that we would incur too high a cost if any of the data was lost.

Step 3. Determine minimum number and type of disks

Now we have to decide what disk type. As the file system size is so big, let us first try the large 18.2 GB Marlin drives. Table 4 on page 80 shows that for the Marlin 18.2 GB disks, when used in RAD5 arrays we can use a GPFS disk throughput rate of 2.89 MB/sec for the full stride RAID write. But 1.44 MB/sec oafish we are doing, we are using the 16 KB non-full stride RAID write. The calculation for Sailfin shows that we, therefore, would need 617/1.44 = 429 data disks. If we use 15+P RAID arrays,

429/15 = 28.6, we need 29 x15+P RAID-5 arrays, therefore, a total of 29 x 16 = 464 Marlin disks.

Step 4. Check and adjust for file system capacity

Using 29 15+P RAID-5 arrays gives us 29 x 15 x 18.2GB = 7.92 TB of file space, much larger than our requirement.

At this point, we should probably go back and use smaller disks or rewrite the application to use 256 KB block writes that will make the RAID-5 arrays perform much faster. If we change to the smaller 4.5 GB Sailfire disks, we can assume the same throughput calculations and end up with a fraction of the file system capacity of 29 x 15 x 4.5 = 1.96 TB of file space.

This is more than we require, but we cannot find smaller disks that are available; so, we will work with this.

Step 5. Determine number of VSD servers required

From Table 5 “Performance of SSA adapters” on page 81, we can see that the GPFS Throughput rate for the 6225 SSA adapter is 60.2 MB/sec for a non-strided read and 14.7 MB/sec for a non-strided write. This now causes our write requirement to become a more stringent requirement.

We would, therefore, require about 532/14.7 = 36 SSA 6225 cards. This gives us a limit of 464/36 = 12.9 disks per adapter before we are adapter bandwidth constrained on the non strided RAID-5 write.

To utilize the maximum bandwidth through the SSA adapter card, we must share equally the disks across the two adapter loops. So, we need to have two RAID-5 arrays, one per loop, as arrays cannot span loops. We also need to try and have the number of SSA disks on one loop divisible by four.

Let us, therefore, try two 5+P RAID-5 arrays per adapter card. You may say that six is not divisible by four, and, of course, you would be right, but by having the loops on one adapter card connected in series with the loops on an adapter card of another node, we can double the number of disks on a loop without increasing the throughput through any one card.

From the previous sections, we know we need at least 429 data disks; so, we would need 429/5 = 85.8 (86)RAID arrays and, therefore, 43 adapter cards. This would mean a total of 43 x 12 = 516 disks. This uses both two loops of each SSA adapter card; so, utilizing the maximum bandwidth from the card.

Now, let us look at the server options. If we choose the POWER3 SMP Wide nodes we can have up to three adapter cards per node. With three cards the total GPFS data throughput on write for the node would be 3 x 14.7 = 44.1MB/sec. From Table 6 on page 82, we can see that for this

node we have a GPFS Switch throughput limit of 94.5 MB/sec; so, this limit is not exceeded.

The number of servers we need is 43/3 = 14.3; so, let us call it 14 servers for the sake of symmetry. This options would give us a required single server GPFS throughput on the switch of 532/14 = 38 MB/sec for write and 617/14 = 44.1 MB/sec for read.

Step 6. Determine VSD server configuration

We now have decided on 14 Power 3 wide nodes each with three 6225 adapter cards each with 12 disks. The total amount of disks is 12 x 3 x 14 = 504. Using the 504 Sailfire disks we have would give us 504/14=36 disks per node or two 5+P RAID-5 arrays per adapter.

We have 36 disks per node using both loops of every 6225 adapter card with each loop having 12 disk connecting two SSA adapters between a pair of nodes (see “Example 2 SSA loop disk layout for a pair of connected nodes” on page 100 for the disk/loop layout). With three adapter cards per node, this gives a total of 36 disks connected between a pair of nodes. This configuration allows the use of the recoverability feature of the RVSD software such that if one node of the pair fails, the remaining node can be set up to take over serving the VSDs owned by the failed node albeit at a reduced data throughput rate as the single SSA adapter throughput would considerably constrain the performance.

For such configurations, where SSA loops are connected between node A and Node B, it is very important to ensure that the disks used by each node have the shortest physical path to the owning node.

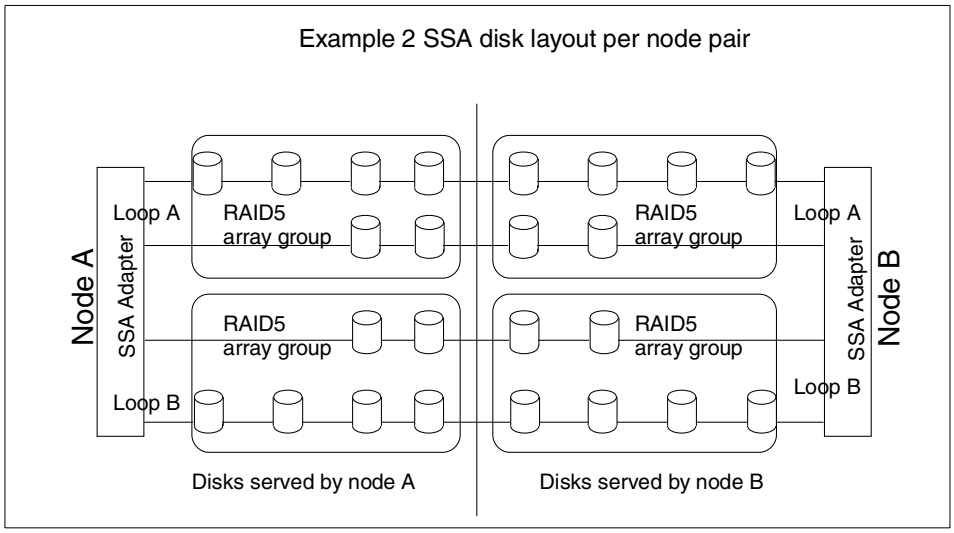


Figure 28. Example 2 SSA loop disk layout for a pair of connected nodes

Chapter 4. Tuning GPFS

GPFS performance depends on the correct specification of its parameters as well as the correct tuning of the function that it uses. Although you can modify your GPFS configuration after it has been set, a little consideration before installation and initial setup will reward you with a more efficient and tuned GPFS file system.

General Parallel File System (GPFS) provides shared access to files across all SP nodes. To achieve this, GPFS exploits a number of facilities provided by the SP, such as Virtual Shared Disk, Group Services, Switch subsystem, and the other components of the AIX operating system.

To achieve optimal performance from GPFS, you must consider the following areas:

Applications

The I/O patterns, block size, files access type, and file size used by the application set are major factors in the determination of GPFS performance. Refer to Chapter 2, “Application considerations” on page 57 for more detailed information.

Network parameters

Tuning the network is critical to maintain peak throughput for network traffic. When tuning an SP network, there are several components to tune. They are the SP switch buffer pools and TCP/IP tunables. Each is described in 4.1.2, “High impact issues” on page 103 for network tuning.

VSD configurations

The main Virtual shared Disk tunable parameters are as follows:

- Virtual Shared Disk cache buffer
- Buddy buffer
- Maximum I/O request size
- Request blocks
- Number of pbufs

These parameters are discussed, along with relevant tuning consideration, in the next sections.

GPFS configurations

When planning for a GPFS system, you must consider a number of general GPFS configuration issues and parameters that affect your GPFS

performance. These parameters include file system block size, the amount of memory allocated for GPFS, possible striping method, I-nodes size, the number of files to cache, and other parameters as discussed in Section 4.2.3.

The first step involves tuning the GPFS. Before thinking about further steps, you need to review the data collected and analyze it. Organize which data you are collecting and when you are collecting it, log the changes you have made, and then repeat the process to be sure you have achieved your goals. These steps should be performed on every node in your environment. Keeping a detailed log of your system before and after any configuration changes could save hours of distress later. Any changes to the environment, whether you are adding more VSD nodes or changing your GPFS configuration, requires a full review of all your system and GPFS parameters.

Tuning the GPFS is a very difficult task. We highly recommend that a change control system be used to track and monitor any changes to the tunables on any part of GPFS including the control workstation.

4.1 Isolating and identifying problems and bottlenecks

This section considers the various performance stress points that can affect the performance of a GPFS file system. The discussion outlines why each stress point is significant and what the effects can be if the tuning parameters affecting these areas are not set appropriately. For each of these parameters, recommendations are given on:

- How to ascertain the current value of the parameter
- What an appropriate setting might be
- How to change the parameter

This section does not provide guidance on how to verify whether or not a given configuration is performing efficiently. For this, the reader is referred to Section 4.3, "Tuning case studies".

It is assumed, in this section, that the GPFS hardware itself has been correctly sized in accordance with the advice given in Chapter 3, “Sizing GPFS” on page 71. It is also assumed that the GPFS, VSD, and AIX software have been properly installed and configured as recommended in the appropriate manuals. We also assume the SP Switch and other IP networks have been given a basic health check. We strongly recommend checking that these areas are configured and functioning correctly before focussing on GPFS specific bottleneck issues.

4.1.1 Working from the dataflow diagram

We refer to the dataflow diagrams presented in 1.2.3, “Potential bottlenecks on writes and reads” on page 15, which summarize the data flow movements for GPFS read and write requests. In consideration of these diagrams, it is possible to identify high, medium, and low performance impact areas in the data flow. We identify these areas in sections 4.1.2, 4.1.3, and 4.1.4 respectively. An assessment is made of the key configuration and tuning parameters that affect each area and what the implications can be if there are performance bottlenecks in the area. For each of the parameters identified, guidance is given on what an appropriate setting for the parameter might be and how to inspect and modify the setting.

Bottleneck areas are considered high, medium, or low impact according to the following guidelines:

- An area is high impact if it can result in data integrity problems or in loss of access to the file system. It is also considered high impact if performance issues in the area can affect GPFS file system performance by more than a factor of two.
- An area is considered medium impact if performance issues in the area can affect GPFS file system performance by more than 25 percent.
- An area is considered low impact if performance issues in the area have negligible affect on GPFS file system performance.

4.1.2 High impact issues

The high impact tuning issues we consider in this section are:

- File system block size
- Number of IBM Virtual Shared Disks per physical disk
- Buddy Buffer considerations
- pbuf considerations
- Request block limit
- Pagepool size
- Switch receive pool and send pool settings
- prefetchThreads setting
- worker1Threads setting
- max_coalesce setting

4.1.2.1 File system block size

As described in Chapter 2, “Application considerations” on page 57, the file system block size should be sized in consideration of the primary I/O application being hosted. A severe mismatch between the two is likely to

result in a performance degradation due to the overheads of performing large numbers of merged write operations.

The current file system block size can be ascertained by executing the `mmlsfs` command. For example:

```
[serv6:/]# dsh -w v06n09 /usr/lpp/mmfs/bin/mmlsfs /dev/adsm -B
v06n09: flag value          description
v06n09: ---- -------------- -----------------------------------------------------
v06n09:  -B  262144         Block size
[serv6:/]#
```

Currently, only three values of GPFS file system block size are supported. These are 16 KB, 64 KB, and 256 KB.

GPFS is optimized towards sequential access of large files. For applications executing I/O in this manner, the maximum value of 256 KB will be optimal. For other I/O patterns, one of the other options may be more appropriate. Knowledge of the I/O patterns of the target application(s) is needed to make an informed decision.

Another factor in the selection of an appropriate file system block size is the number and size of the applications' data files. The 16 KB setting is recommended for large numbers of small files and 256 KB for fewer numbers of large files. Where applications use both small and large files, a value of 64 KB might prove the best compromise.

File system block size is a parameter that may not be dynamically reconfigured. To change it means rebuilding the file system. It is, therefore, worth considering this parameter carefully before making an initial configuration.

Please refer to Chapter 2, "Application considerations" on page 57 for more information regarding the interfaces between GPFS and applications.

4.1.2.2 Number of IBM Virtual Shared Disks per physical disk

Having too few VSDs configured for the number of physical disks connected to a system can seriously impact performance. This is because the degree of parallelism within the file system will also be reduced. On the other hand, having multiple VSDs per physical disk can also seriously impact performance because of the increased access contention on each disk. Multiple VSDs per physical disk will also make the environment far more complex to manage.

The `lsvsd -l` command can be used to ascertain what VSDs have been configured on a specific node.

As a general guideline, we recommend that one, and only one, physical disk is associated with each IBM Virtual Shared Disk. Configuring one VSD per hard disk in this way optimizes GPFS performance. The `mmcrfs` command for creating GPFS file systems by default creates one IBM Virtual Shared Disk for each physical disk specified for the file system.

It is possible to override the `mmcrfs` default, but only by manually creating the VSD and then passing the name of the VSD that was created to the `mmcrfs` command. This will obviously complicate the overall configuration process. Creating the VSDs manually with the `createvsd` command will allow specific VSDs to be created and assigned to individual hard disks on each node. So, for example, it would be possible to create one VSD per node for the entire bank of local disks that are physically connected to that node.

In the event that you do decided to implement a non default hdisk and VSD configuration, please refer to *Managing Shared Disks,* SC23-4839, for more information.

Please note that changing the number of VSDs per physical disk for an existing GPFS file system requires a complete rebuild of the file system.

4.1.2.3 Buddy buffer considerations

In terms of tuning buddy buffers for GPFS performance, there are two main considerations:

- The maximum size of a buddy buffer
- The maximum number of buddy buffers

The current setting for these parameters can be determined by executing the `vsdatalst -n` command, for example:

```
# vsdatalst -n
VSD Node Information
                                          Initial Maximum     VSD      rw      Buddy Buffer
  node                VSD      IP packet   cache   cache request request minimum maximum size: #
number host_name       adapter    size    buffers buffers   count   count    size    size maxbufs
------ --------------- -------- --------- ------- ------- ------- ------- ------- -------
-------
     1 v06n01i         css0         61440      64     256     256      48    4096  262144       33
     3 v06n03          css0         61440      64     256     256      48    4096  262144       33
     5 v06n05          css0         61440      64     256     256      48    4096  262144       33
     7 v06n07          css0         61440      64     256     256      48    4096  262144       33
     9 v06n09          css0         61440      64     256     256      48    4096  262144        2
    11 v06n11          css0         61440      64     256     256      48    4096  262144        2
    13 v06n13          css0         61440      64     256     256      48    4096  262144        2
    15 v06n15          css0         61440      64     256     256      48    4096  262144        2
```

The command returns virtual shared disk node information for all nodes in the current SP partition. From the above table, we can see the maximum buddy buffer size is 262144 for all nodes. The number of maximally sized buddy buffers is 33 on v06n01, v06n03, v06n05, and v06n07. In this example, these are the VSD server nodes. On the remaining nodes, which are GPFS nodes, the maximum number of buddy buffers is set to 2.

Both the minimum and maximum buddy buffer sizes should be specified as a power of 2 because buddy buffers are allocated in powers of 2. The default values are 4 KB and 64 KB, respectively. For a given I/O request, a buffer will be sized to the first power of 2 that is larger than the request. For this reason, there is no sense making the minimum value lower than 4 KB, as only one I/O request can be pending for a page. 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.

The maximum buddy buffer size should be set to the block size of the largest GPFS file system served in the SP partition. Having the maximum size set lower than this will result in extra overhead in 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 may result in over allocating memory to buddy buffer space. We recommend setting the minimum and maximum buddy buffer sizes the same on VSD servers and clients.

On VSD clients, we recommend setting the maximum number of buddy buffers to one. For servers, a guideline minimum setting for this parameter is four times the number of disks attached to an individual VSD server. Higher

values than this may be required if the application I/O patterns are not uniform. Four times the number of disks attached to a VSD server will be more effective in most situations. Having an insufficient value can seriously degrade performance because requests for buddy buffers will become queued. A more precise way of estimating this parameter can be used if you know the remote I/O throughput on the VSD server and the average turn around time of an individual request. For example, if the throughput is 10 MB/s and, the turn around time is 60 ms, a minimum 0.6 MB of buddy buffer storage would be required. This value should then be at least tripled to provide an adequate safety margin. Always monitor for retries using the `statvsd` command as shown in 4.2.1.2, “Analyzing IBM Virtual Shared Disks” on page 139.

If the virtual shared disk statistics consistently show requests queued waiting for buddy buffers, do not consider buddy buffers in isolation. Consider also increasing the size of the switch send pool or spreading the data over disks attached to other nodes. However, increasing the switch pools size will not solve the problem if this is caused by slow disks.

The values of the parameters discussed in this section can be set either by using the IBM Virtual Shared Disk Perspective graphical user interface or by use of the `updatevsdnode` command. To change the maximum number of buddy buffers, for example:

```
# updatevsdnode -n 1 -s 33
```

To change the maximum size of a buddy buffer:

```
# updatevsdnode -n 1 -x 262144
```

In this example, the maximum buddy buffer size on node 1 would be set to 256 Kb.

Please note that these changes are not dynamic. The `updatevsdnode` command itself only updates the affected parameters in the SDR. To make the configuration changes active, the GPFS file system must be unmounted, the gpfs daemons stopped and the `ha_vsd reset` command issued before bringing the file system back on line.

4.1.2.4 pbuf considerations

The number of configured pbufs can have a critical affect on GPFS and overall system performance. Pbufs provide a means of controlling the number of pending logical volume requests for each VSD device at its server node. Pbufs are also known as read/write request control blocks. Too few pbufs could impair performance. Bear in mind, however, that every VSD device is

allocated a pbuf at its server and at every client. Because pbufs are allocated out of pinned kernel memory, it is easy to see that configuring too many of them could seriously impact overall system performance of VSD servers and clients and could lead to eventual exhaustion of the kernel heap with a resulting system crash.

Each pbuf consumes 128 bytes of kernel memory, and this memory is pre-allocated and dedicated for each VSD device until the VSD is unconfigured. Current levels of AIX limit the total kernel heap space to one segment of 256 MB; so, caution is recommended to not set this parameter too high although in AIX 4.3.1 and later the network memory has been moved to a separate segment. The default setting is 48 per device.

The current number of pbufs for each VSD device on a node can be identified across the whole SP partition from the `vsdatalst -n` command output under the rw request count column. For example:

```
[v06n05:/]# vsdatalst -n
      VSD Node Information
                                          Initial Maximum     VSD      rw      Buddy Buffer
  node                 VSD      IP packet   cache   cache request request minimum maximum size: #
number host_name       adapter    size    buffers buffers   count   count    size    size maxbufs
------ --------------- -------- --------- ------- ------- ------- ------- ------- ------- -------
     1 v06n01i         css0        61440      64     256     256      48    4096  262144      33
     3 v06n03          css0        61440      64     256     256      48    4096  262144      33
     5 v06n05          css0        61440      64     256     256      48    4096  262144      33
     7 v06n07          css0        61440      64     256     256      48    4096  262144      33
     9 v06n09          css0        61440      64     256     256      48    4096  262144       2
    11 v06n11          css0        61440      64     256     256      48    4096  262144       2
    13 v06n13          css0        61440      64     256     256      48    4096  262144       2
    15 v06n15          css0        61440      64     256     256      48    4096  262144       2
[v06n05:/]#
```

In this instance, we can see that there are 48 pbufs allocated for each VSD device in all nodes of the partition. This means that if one of these nodes was a server for 64 VSD devices, then on that node there would be 48 * 64 pbufs allocated for this purpose. This would constitute just over 3 MB of pinned kernel memory.

As a guideline, we recommend that the number of pbufs is set to approximately 16 per VSD device. This will allow the queuing of reads at the VSD server. In the situation that there is evidence of a pbuf shortage, increasing the number of pbufs above this parameter does not imply

performance will be improved. It is more likely the bottleneck will be moved to other parts of the system.

Before resetting the number of pbufs, we strongly recommend reviewing the memory demands this will impose on the kernel. The following formula determines how much of the kernel heap is dedicated to pbuf storage:

memory_allocated = num_vsds * num_pbufs * 128

where:

- num_vsds is the number of IBM Virtual Shared Disks configured
- num_pbufs is the rw_request_count.

A good way to check the planned number of pbufs is within range is to assess the amount of available kernel memory and then subtract from it the memory_allocated value determined above. The amount of available kernel heap space can be assessed from the `xmalloc -u` subcommand of the `crash` utility. For example:

```
[v06n05:/]# crash
WARNING: Using crash on a live system can potentially
         cause a system crash and/or data corruption.
> xmalloc -u
Kernel heap usage
    Storage area: 0x50000000..0x5ffbefff (268169216 bytes, 65471 pages)
    Primary heap allocated size: 58849440 (57470k)
    Alternate heap allocated size: (14310k)

Overflow heap usage
    Storage area: 0x11b5f48..0xffc3f47 (249618432 bytes, 60942 pages)
    Primary heap allocated size: 67584896 (66000k)
    Alternate heap allocated size: 0 (0k)

> q
[v06n05:/]#
```

In this example, we can see that the amount of available kernel heap is (268169216 - 58849440 - 14654080) / (1024*1024) bytes, which is approximately 185 MB.

An an example, if we wished to limit 16 MB of the kernel heap for pbufs, and 1300 IBM VSDs are configured, nreq should not be greater than 100.

The number of pbufs to be allocated per virtual shared disk can be modified by the rw_request_count parameter of the `updatevsdnode` command or through the SMIT vsdnode_dialog fast path.

For example, to set the number of pbufs on node 1 to 48:

```
# updatevsdnode -n 1 -p 48
```

This change is not dynamic. The `updatevsd` command itself only updates the affected parameters in the SDR. To make the configuration changes active, the GPFS file system must be unmounted, the gpfs daemons stopped, and the `ha_vsd reset` command issued before bringing the file system back on line.

4.1.2.5 Request block limit

The maximum number of request blocks limits the number of possible outstanding VSD requests at a particular node. It is a high impact VSD parameter as it limits in particular the number of VSD requests that can be pending from individual VSD clients. Specifying an inordinately high value could have a serious performance impact. A large number of request blocks could result in flooding of the network, resulting in servers running out of mbufs and having unnecessary retransmissions. What constitutes a large number of requests depends on how large the average request size is and how many nodes there are in the system. If the number is too small, local requests can queue up waiting for a request block to become available. Please note that this parameter is a *per node* limitation, unlike the pbufs parameter, which is a *per VSD* limitation.

The current setting for the number of request blocks can be ascertained with the smitty fast path:

```
# smitty vsd_mgmt
```

and then following the menu items Set/Show VSD Device Driver Optional Parameters and Show VSD Device Driver Optional Parameters. Alternatively, use the command `vsdatalst -n` and check the "VSD request count" column. In this example, we can see the number of request blocks is 256 for all nodes in the partition:

```
[v06n05:/]# vsdatalst -n
      VSD Node Information
                                        Initial Maximum     VSD      rw      Buddy Buffer
  node                VSD      IP packet   cache   cache request request minimum maximum size: #
number host_name       adapter    size    buffers buffers   count   count    size    size maxbufs
------ --------------- -------- --------- ------- ------- ------- ------- ------- ------- -------
     1 v06n01i         css0        61440      64     256     256      48    4096  262144      33
     3 v06n03          css0        61440      64     256     256      48    4096  262144      33
     5 v06n05          css0        61440      64     256     256      48    4096  262144      33
     7 v06n07          css0        61440      64     256     256      48    4096  262144      33
     9 v06n09          css0        61440      64     256     256      48    4096  262144       2
    11 v06n11          css0        61440      64     256     256      48    4096  262144       2
    13 v06n13          css0        61440      64     256     256      48    4096  262144       2
    15 v06n15          css0        61440      64     256     256      48    4096  262144       2
[v06n05:/]#
```

The default setting for the number of request blocks is 256. (The size of each request block is approximately 76 bytes.)

Because large requests may be broken up into smaller sub-requests, the number of outstanding VSD requests may be several times greater than the total number of pending read/write requests.

Although the `statvsd` command reports the number of times there is no request block available, queueing for request blocks does not necessarily imply a performance bottleneck. If you increased the number or request blocks infinitely, queueing would occur elsewhere in the operating system.

As a general guideline, we recommend setting this parameter to 128. A more accurate estimate can be derived by considering:

- Average rate of issue of I/O operations
- Average number of sub-requests
- Average I/O request response time

If these values on a particular node were, 1000 I/Os per sec, 3 and 50mS respectively, then a minimum of 150 (1000 * 3 * 0.05) request blocks would be required. In practice, the parameter would be set over this level to allow for a degree of contingency. When allowing for this contingency, consider the frequency, severity, and duration of I/O bursts from the application.

The number of request blocks can be set and changed with the IBM Virtual Shared Disk Perspective graphical user interface or the `vsdnode` and

updatevsdnode commands, respectively. For example, to change the number of request blocks to 128 on node 1, the following would be used:

```
# updatevsdnode -n 1 -r 128
```

This change is not dynamic. The updatevsd command itself only updates the affected parameters in the SDR. To make the configuration changes active, the gpfs file system must be unmounted and the gpfs daemons stopped. Once the GPFS daemons are stopped, you must also explicitly stop RVSD with the ha.vsd stop command. Then you must explicitly unload the VSD device driver by issuing ucfgvsd -a. Then, you can finally issue ha_vsd reset.

4.1.2.6 Pagepool size

The GPFS pagepool is a high impact performance area because it is the pagepool mechanism that allows GPFS to implement read and write requests asynchronously via read ahead and write behind mechanisms. It also allows *re-use* of current data already present in the pagepool area analogous to the JFS buffer cache. However, unlike JFS, the GPFS pagepool is pinned memory; so, care needs to be taken with sizing it. Setting pagepool too high could impact overall systems performance because of the reduced real memory available to applications. Pagepool memory is allocated within the mmfs daemon's address space. The mmfs daemon itself will fail on start-up with an Abnormal exit error if it cannot successfully pin the specified amount of pagepool memory. On the other hand, setting pagepool too low could seriously degrade the performance of certain types of GPFS applications.

The pagepool parameter can be set to between 4 MB and 512 MB per node. The default setting is 20 MB. The current setting of the pagepool parameter can be determined by inspecting the file /var/mmfs/etc/mmfs.cfg. For example:

```
#[v06n01i:/]# grep "^pagepool" /var/mmfs/etc/mmfs.cfg
pagepool 20M
[v06n01i:/]#
```

If GPFS itself determines the pagepool to be very insufficient, error messages will be written to the mmfsd log file /var/adm/ras/mmfs.date. If such messages are being written, and the problem is not addressed, the eventual result would be a major slowdown but not a loss of file system integrity. These messages are generated when it becomes necessary to steal more than a pre-defined percentage of buffers from the pagepool area for re-use consistently over a set number of sync periods. More precise information on the rate of pagepool buffer stealing can be ascertained from the mmfsadm command. For example:

```
# mmfsadm dump pgalloc
```

```
Statistics:
Total number of buffer allocations:           39
Number of calls to steal a buffer:             1  =   3% of allocations
Number buffers searched:                       0  = 0.0 searches per steal
Total number of buffers:                      17
Number of calls to assign buffers:             2
Number of times initial agg > 0:               0  =   0% of steals
```

The optimum setting for pagepool depends on the rate at which the file system is servicing I/O requests and the I/O pattern of the applications driving those requests. Increasing pagepool will increase the amount of cached data available to applications. This will provide performance benefits in applications that do large amounts of I/O quickly and in applications where reuse of data is high. A sufficiently large pagepool is also important 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 may significantly improve performance. The setting of pagepool can also be particularly critical for applications that 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.

Consider an example where an application periodically performs high and intense levels of I/O. For simplicity, we assume that this is the only application driving the file system on the particular GPFS node. It is important to have an assessment of:

- The rate of bursts of I/O from the application
- The amount of data to be transferred in one burst
- The rate at which the application can deliver data for read/write I/O
- The rate at which data can be written from the pagepool out to disk

Let us assume that the application writes bursts of data where each burst is 800 MB. Also, let us assume the application can deliver this for writing at the rate of 40 MB/sec. In a real situation, data would be flowing over the SP Switch into a VSD server from which it would be reassembled into a buddy buffer and flushed out to disk. We assume that the disks are the bottleneck in this process. Assume that each disk can deliver data at 5 MB/sec and there are four disks configured in the file system.

Assuming an even distribution of data, this implies an aggregate throughput of 20 MB/sec. For the application to execute unheeded for the first complete

burst, that is, without having to wait for any pagepool space to be freed, a pagepool of 800 MB would be required. This would allow the whole set of data being written to be buffered. It would take 20 seconds for this buffer to fill (800 MB at 40 MB/sec) and a further 20 seconds for it to empty once it has become full (400 MB at 20 MB/sec). However, the buffer starts emptying as soon as the block fills so the total timearound for the applications is 20 + n sec, where n is a function of the disk speed and the exact application pattern; so, the application cannot generate data faster than the disks can accept it on a sustained basis.

The pagepool size may be reconfigured dynamically with the `mmchconfig -i` command. For example:

```
# mmchconfig -l pagepool=60M v06n09 -i
```

Please note the value must be suffixed with the character M.

The -i flag was introduced with GPFS V1.2. On earlier revisions of GPFS, the command should be entered without -i flag and the GPFS daemons restarted on the target nodes for the change to take effect.

Because the setting of the pagepool parameter is so specific to the amount of I/O requests being generated from a node and the application I/O pattern, this is one parameter that it is worthwhile assessing on a per node basis. Further, because it can be modified dynamically from GPFS V1.2 onwards, 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 to increase the pagepool and reset it again wrapped around the main application command script.

Another point to be bear in mind is that because this is a dynamic parameter, 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.

4.1.2.7 Switch receive pool and send pool settings

Both of these parameters are high impact because, if they are not set correctly, GPFS performance can be severely impacted.

The `lsattr` command can be used to check the current settings of send pool and receive pool buffer space on a specific node. For example:

```
[v06n01i:/]# lsattr -E -l css0
kernel_memory  0xf1000000 TB3MX kernel memory address False
user_memory    0xf2000000 TB3MX user memory address   False
int_priority   3          Interrupt priority          False
```

```
int_level      14         Bus interrupt level         False
spoolsize      16777216   Size of IP send buffer      True
rpoolsize      16777216   Size of IP receive buffer   True
adapter_status css_ready  Configuration status        False
[v06n01i:/]#
```

It is critical that the receive pool parameter is set to the maximum of 16 MB on virtual shared disk servers. If it is not set to the maximum on VSD servers, it is very likely the result will be dropped packets and retries at higher protocol levels. As a guideline, we also recommend setting the receive pool to the maximum on GPFS nodes unless there is a particular shortage of memory, and there are no VSD servers configured on the GPFS nodes.

Similar arguments hold true for the send pool parameter. It is critical that the send pool is set to the maximum of 16 MB on VSD servers. If it is not set to the maximum on VSD servers, the outcome may be a shortage of send pool space with buddy buffers consequently being tied up with the data. This could result in the situation where disks become idle despite a backlog of pending requests. We also recommend setting the send pool to the maximum on GPFS nodes unless there is a particular shortage of memory. In such situations, consideration could be given to reducing it providing either the I/O rates are low, or the read to write ratio is very high.

The settings of send pool and receive pool can be changed by use of the `chugs` command. For example:

```
/usr/lpp/ssp/css/chgcss -l css0 -a rpoolsize=16777216 -a spoolsize=16777216
```

Please note that these messages that are returned from the `chgcss` command can be ignored:

```
There was customized rpoolsize, and new value !=default
There was customized spoolsize, and new value !=default
```

The affected nodes must be rebooted for these changes to take effect.

4.1.2.8 prefetchThreads parameter

We consider prefetchThreads to be a high impact parameter because it exercises direct control over the number of threads used for sequential read I/O.

The minimum value for prefetchThreads is 1, and the maximum, which is the default, is 48. The current setting of the prefetchThreads can be determined by inspecting the file /var/mmfs/etc/mmfs.cfg. For example:

```
#[v06n01i:/]# grep "^prefetchThreads" /var/mmfs/etc/mmfs.cfg
prefetchThreads 48
```

```
[v06n01i:/]#
```

This parameter is specific to a GPFS node; so, there is no reason why it cannot be set to different values on different nodes if this is found to be appropriate to the application environment. Remember though, that GPFS will only activate read ahead processing if it determines from the I/O pattern of the application that it would be beneficial to do so.

The arguments for setting this parameter relate closely to those for worker1Threads. Increasing the setting can increase the degree to which threads can retrieve data for the same request concurrently. It should not be assumed that the maximum value will always optimize performance as the various I/O sub-systems may not be able to deliver sufficiently fast to the individual threads. In fact, setting prefetchThreads too high could result in the I/O sub-systems being over stressed with the consequent failure of the GPFS file system and a device unavailable error.

The best way to assess the optimum number for prefetchThreads is to consider the total available VSD server bandwidth for read I/O and to compare this against a realized value for the rate at which a single thread can retrieve data from the VSD sub-system. To assess the bandwidth of a single thread, one could set prefetchThreads very low, say to 2, and compare a run for this setting to a run with prefetchThreads increased by one to a total value of 3.

Consider, for example, that four disks are configured on the VSD server and each has a sustainable I/O read throughput of 5 MB/s. We would then conclude that an evenly striped read could be read off the disks at the rate of 20 MB/s. If we determined from our thread test that an individual read thread can process data at the rate of 3 MB/s, then an optimum number of pre-fetch threads for this process would be seven.

The value of prefetchThreads applies to all read requests being performed on the GPFS node for all GPFS file systems being accessed from that node. The number determined, therefore, needs to be scaled according to the number of concurrent applications accessing GPFS file systems from that node.

During the tests that were performed during the course of writing this redbook, it was noticed that the mmfs daemons, which control the allocation of read ahead threads, exercise a further degree of control over thread allocation over and above the setting of prefetchThreads. So, even if an application is making a very high number of read requests, mmfsd will not necessarily initialize prefetchThread number of threads.

The prefetchThreads parameter cannot be modified dynamically or with the `mmchconfig` command. The mmfs configuration file mmsdrcfg1 must be manually retrieved from the SDR and edited to update the parameter. The file should then be returned to the SDR, and GPFS should be restarted on the affected nodes. Here is an example of the sequence:

```
# cd /tmp
# SDRRetrieveFile mmsdrcfg1 mmsdrcfg1
... Edit the file to change worker1Threads as required
# SDRReplaceFile /tmp/mmsdrcfg1 mmsdrcfg1
# stopsrc -c -s mmfs
# startsrc -s mmfs
```

4.1.2.9 worker1Threads setting

This parameter is high impact because it exerts direct control over the number of threads used by the mmfs daemon to exercise sequential write behind. Setting this parameter too low may impair the ability of GPFS to implement write behind sufficiently fast for the application and thus deprive it of write I/O bandwidth. On the other hand, setting it too high could in some situations over stress the I/O sub-systems and cause I/O requests to stall, which could possibly eventually result in device access errors to the file system. The optimum setting for this parameter will depend both on the I/O pattern of the target application and the I/O sub-systems used in the SP partition.

The minimum value for worker1Threads is 1 and the maximum, which is the default, is 72. The current setting of worker1Threads parameter can be determined by inspecting the file `/var/mmfs/etc/mmfs.cfg`. For example:

```
#[v06n01i:/]# grep "^worker1Threads" /var/mmfs/etc/mmfs.cfg
worker1Threads 72
[v06n01i:/]#
```

The worker1Threads parameter is node specific, so can be set to different values on different nodes if appropriate to the application environment.

The comments in the mmfs configuration file recommend setting worker1Threads according to the prefetchThreads parameter:

worker1Threads = 1.5 * prefetchThreads

Whilst this is a reasonable general guideline, we would also recommend that the total available VSD server bandwidth for write I/O is compared against a realized value for the rate at which a single thread can place data into the VSD sub-system. The argument is identical to that for prefetchThreads.

Increasing worker1Threads from a low value increases the degree to which threads can write data for the same I/O request concurrently. This does not imply that the maximum value will always optimize performance as the various I/O sub-systems may not be able to process data sufficiently fast from the individual threads. As with prefetchThreads, setting worker1Threads too high may over-stress the VSD server I/O sub-systems with the risk of loss of file system availability.

In the situation that I/O sub-systems are severely over-stretched by write I/O requests, I/O requests may stall with a potential for the file system to go off line. In this situation, lowering the worker1Threads parameter is an effective means to bringing the write I/O levels back down to a manageable level. The other options here might be to reduce the setting of the pagepool parameter or the VSD request blocks parameter, but neither of these two options are ideal. Reducing pagepool increases the risk of the system thrashing, while reducing the request blocks parameter would impact metadata traffic as well, which could, in turn, lead to deadlock situations.

As with the prefetchThreads parameter, it should be noted that the mmfs daemon exercises its own control on the number of write behind threads over and above the setting of worker1Threads. This means that even after you have sized worker1Threads correctly, the daemon may limit the active number of threads well below the maximum you have specified.

The worker1Threads parameter is modified by extracting the mmsdrcfg1 configuration file from the SDR and editing it. The file should then be returned to the SDR, and GPFS should be restarted on the affected nodes. Please note that this parameter cannot be modified dynamically with the `mmchconfig` command. Please refer to 4.1.2.8 on page 115 for more details of the procedure.

4.1.2.10 max_coalesce setting

This is a high impact parameter for sites that are using SSA RAID 5 disk arrays in certain RAID configurations. If the value is set too low, then write performance can be seriously impacted.

The current value of max_coalesce can be ascertained via the `lsattr` command or the smitty fastpath `smitty chgssardsk`. For example:

```
[v06n05:/]# lsattr -El hdisk4
pvid            00008655ca50d3bf0000000000000000 Physical volume identifier  False
queue_depth     48                               Queue depth                 True
write_queue_mod 0                                Write queue depth modifier  True
adapter_a       ssa0                             Adapter connection          False
adapter_b       none                             Adapter connection          False
primary_adapter adapter_a                        Primary adapter             True
reserve_lock    yes                              RESERVE device on open      True
connwhere_shad  8681374B12BE4CK                  SSA Connection Location     False
max_coalesce    0x1e0000                         Maximum coalesced operation True
size_in_mb      67683                            Size in Megabytes           False
location                                         Location Label              True
[v06n05:/]#
```

The default value of max_coalesce for an SSA logical disk is 0x20000 (128KB) for each data disk configured into the array. So for a 4+P array, the default setting of max_coalesce would be 0x80000 (512 KB).

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 this end, it is recommended that, where possible, the setting of max_coalesce matches with the array stripe size and the GPFS file system block size. For a GPFS file system configured with a 256 KB block size, a recommended setting for max_coalesce on a 4+P raid array would, therefore, be 0x40000, that is, 256 KB. Setting max_coalesce so that it is smaller than the GPFS file system block size means that I/O operations will always have to initiate several I/Os across the array to read or write a single file system block. On the other hand, setting max_coalesce greater than the GPFS file system block size may also result in degraded write performance. This is because the SSA adapter will incur additional overhead as it will have to perform modified writes for a single I/O operation in order to compute parity.

For other RAID configurations, the options are not so straight forward. For a 7+P configuration, for example, a full stride RAID write would comprise 448 KB of data, which does not match onto any of the supported GPFS file system sizes. So, in this situation, it would not be possible to set max_coalesce as optimally for GPFS file systems as is possible for 4+P. This was stressed out by the tuning tests where it was observed that tuning

max_coalesce on 15+P raid configurations, for example, had little effect whatever the setting. On the other hand, setting max_coalesce to 256 KB on a 4+P RAID-5 array with an application writing at 256 KB buffers, we found write performance was approximately twice than when max_coalesce was set to 128 KB. Interestingly, write performance did not degrade noticeably from the 256 KB setting if max_coalesce was further increased to 1020 KB. In this situation, we would, nevertheless, recommend sticking to the full stride RAID value of 256 KB.

Note that the value for max_coalesce should not currently exceed 0xFF000 (1020 KB) even though the SSA device driver itself allows values up to 2 MB. Values in excess of 0xFF000 may result in I/O operations being rejected. This should be remembered when configuring 15+P RAID-5 arrays as the default setting will be greater than 1020 KB.

The max_coalesce can be modified by use of the `chdev` command or the SMIT fastpath `chgssardsk`. It is first necessary to unmount the gpfs file system. The gpfs volume group must be varied off, and the logical device removed before making the change. For example:

```
# varyoffvg gpfs4
# rmdev -l hdisk4
# chdev  -l hdisk4 -a max_coalesce=262144
# mkdev -l hdisk4
# varyonvg gpfs4
```

One final point to make here is that if a RAID 5 configuration is being considered for the first time alongside a new GPFS file system, then it is a very good idea to carefully consider the application I/O pattern against the possible settings of the file system block size and the potential RAID stripe width. If a suitable match can be found so that max_coalesce balances with the application buffer size and the file system block size, and it can also be set so that it equates to a full stride of the RAID array, then write performance will be significantly enhanced.

4.1.3 Medium impact issues

This section considers medium impact tuning issues. These are issues that can generally affect GPFS file system performance by more than 25 percent. The areas we look at are:

- Dedicating VSD servers
- Splitting data and metadata across VSDs
- max_IP_msg_size setting
- ipqmaxlen setting
- Number of GPFS nodes

- maxFilesToCache setting
- mallocsize setting
- worker2Threads setting
- thewall
- Other TCP/IP parameters

4.1.3.1 Dedicating VSD servers

The issue of whether or not to dedicate VSD servers is of medium impact in respect to tuning GPFS. VSD servers do not have to be dedicated. GPFS applications can run on the same nodes as the VSD servers themselves. In the situation where there is no shortage of node capacity, we would recommend dedicating the servers. Where this is not practical, a local decision will have to be made as to how the application nodes and VSD servers are partitioned.

There are two factors to bear in mind:

- Dedicated VSD servers reduce the risk of potential CPU and I/O bottlenecks that could otherwise be incurred by time sharing with other applications. It will also minimizes the risk of critical GPFS file system bottlenecks that could arise from one or more particular VSD servers being more heavily loaded than others.
- In situations where it is not possible to dedicate VSD servers, it is recommended that only applications that are not time critical are run on those nodes.

4.1.3.2 Splitting data and metadata across VSDs

It is possible to nominate individual VSDs within a particular GPFS file system to contain data only, metadata only, or a combination of both. The default is to split data and metadata across all VSDs.

It might be considered that there are situations where this default behavior could be changed to an advantage. Metadata traffic tends to consist of relatively small random reads and writes, and, therefore, is quite different to the large sequential I/O of the typical GPFS application. Further, splitting data from metadata might also be considered advantageous when storing application data on a raid array. In these situations, one might think at first to store metadata on different VSDs. But one has also to think of the total I/O bandwidth considerations. If metadata is isolated on a few separate VSDs, then that could mean access to those VSDs rapidly becomes a performance bottleneck itself. We believe that separating data from metadata is only likely to improve performance in the situation where the same number of disks are used for metadata as data. This is clearly 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 replicated (we recommend you always replicate metadata). The advantages of storing data on RAID arrays configured with hot-spare standbys will be compromised if they rely on non-RAID disks for storing metadata.

In light of these considerations, we recommend adopting a general guideline of interspersing data and metadata across VSDs and always replicate metadata.

For more information on metadata management performance, refer to 6.3, “Metadata tests” on page 207.

4.1.3.3 max_IP_msg_size setting

This is a medium impact parameter. It defines the largest size of the packets that the virtual shared disk software will send between the client and the server. Its value can vary between 512 bytes and 65024 bytes (63.5 KB). The default setting is 61440 bytes. The current setting of the parameter can be ascertained from the `vsdatalst -n` command and inspecting the column IP packet size. For example:

```
[v06n01i:/]# vsdatalst -n
       VSD Node Information
                                          Initial Maximum     VSD      rw      Buddy Buffer
  node                VSD      IP packet   cache   cache request request minimum maximum size: #
number host_name       adapter    size    buffers buffers   count   count    size    size maxbufs
------ --------------- -------- --------- ------- ------- ------- ------- ------- ------- -------
     1 v06n01i         css0         61440      64     256     256      48    4096  262144      33
     3 v06n03          css0         61440      64     256     256      48    4096  262144      33
     5 v06n05          css0         61440      64     256     256      48    4096  262144      33
     7 v06n07          css0         61440      64     256     256      48    4096  262144      33
     9 v06n09          css0         61440      64     256     256      48    4096  262144       2
    11 v06n11          css0         61440      64     256     256      48    4096  262144       2
    13 v06n13          css0         61440      64     256     256      48    4096  262144       2
    15 v06n15          css0         61440      64     256     256      48    4096  262144       2
You have mail in /usr/spool/mail/root
[v06n01i:/]#
```

Larger values of max_IP_msg_size will result in fewer packets being required to transfer the same amount of data. In the context of using GPFS with VSDs configured over the SP Switch, we recommend setting max_IP_msg_size to the maximum value, 65024 bytes.

The max_IP_msg_size parameter can be modified by use of the `ctlvsd` command. For example:

```
# ctlvsd -r 1 -M 61440
```

This would set max_IP_msg_size to 60 KB on node 1.

4.1.3.4 ipqmaxlen setting

This parameter is medium impact. If it is set too low, the result can be dropped IP packets and consequent performance degradation. Because both GPFS and VSD software make extensive use of the IP protocol, the default setting of 128 is often insufficient. Setting ipqmaxlen too high will not result in a loss of memory but may result in a CPU overhead because of the additional processing required for the extended number of IP packets.

The current setting of the ipqmaxlen parameter can be displayed by use of the `no` command. For example:

```
[v06n01i:/]# no -o ipqmaxlen
ipqmaxlen = 100
[v06n01i:/]#
```

We recommend, as a general guideline, that ipqmaxlen should be increased to 512 on both VSD servers and GPFS nodes. Increases above this size should be made experimentally in relatively small steps, such as 128.

The value of the ipqmaxlen can also be modified by use of the `no` command. For example:

```
# /usr/sbin/no -o ipqmaxlen=newvalue
```

Please note that this parameter can be changed dynamically and is effective until the next reboot of the node. The change should also be incorporated into the /etc/rc.net start-up file to make it last across reboots.

Please refer to Section 4.1.3.9 “thewall” on page 127 and Section 4.1.3.10 “Other TCP/IP parameters” on page 127 for information on other network related parameters.

4.1.3.5 Number of GPFS nodes

In a large SP system, there may be spare nodes available that could be used to increase the number of nodes on which the GPFS application runs. However, it does not follow automatically that increasing the number of nodes will guarantee an improvement in overall performance. In addition, there may be other constraints which fix the number of nodes on which the application can run.

Increasing the number of GPFS nodes will increase overall availability because it will be easier to meet GPFS quorum requirements. For example, an environment of two GPFS nodes necessitates, by default, that both systems are up and running before GPFS can start. An environment with eight GPFS nodes requires five, meaning that the application can still be made available even if three nodes fail to start up properly.

Whether or not increasing the number of nodes improves performance of the GPFS application will depend on the following:

1. Whether the execution of the application is locally bottlenecked on the GPFS nodes. If it is the GPFS or VSD sub-system, which is the primary bottleneck, then clearly these should be addressed first.
2. Whether the application architecture readily permits segmentation of the non-GPFS part of a single execution to run in serial on multiple machines or in parallel on multiple machines.

Increasing the number of nodes may be worthwhile where the application is locally bottle-necked, but the GPFS and VSD sub-systems are not, and where the non-GPFS processing can be distributed across multiple nodes.

Please also refer to Section 3.2.4 on page 90 for information regarding sizing the number of gpfs nodes to the number of VSD servers.

4.1.3.6 maxFilesToCache

The maxFilesToCache is considered a medium impact parameter. 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. This parameter is closely related to the mallocsize parameter, and we recommend changes to the two are made in tandem. The default setting for maxFilesToCache is 200. The current setting of maxFilesToCache on an individual node can be determined by inspecting the file /var/mmfs/etc/mmfs.cfg. For example:

```
[v06n09:/var/mmfs]# grep "^max" /var/mmfs/etc/mmfs.cfg
maxFilesToCache 200
[v06n09:/var/mmfs]#
```

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 the particular node. If a specific node is expected to open more than 200 files, then consideration should be given to increasing the parameter accordingly. When only a few files need to be accessed concurrently, the value of maxFilesToCache can be reduced so that the overhead of maintaining unrequired internal memory buffers is minimized.

Please note that whatever the setting of maxFilesToCache, GPFS will not use more than 50 percent of the mallocpool for caching i-nodes. We recommend maxFilesToCache and mallocsize are both set on a per node basis as required.

The maxFilesToCache can be modified with the `mmchconfig` command. For example:

```
# mmchconfig maxFilesToCache=300 v06n01i
```

This would change the maxFilesToCache parameter to 300 on node v06n01i. Please note that this change is not dynamic. The GPFS file system must be unmounted, the mmfs daemon restarted and the file system re-mounted on the target nodes for the change to take effect. Nevertheless, for applications that make dedicated use of the GPFS file system, commands to reset maxFilesToCache could be wrapped around an application execution script.

Version 1.1 of GPFS imposes a default limit of 100 on maxFilesToCache. We recommend this value not be increased if the GPFS node is running under AIX 4.2.1 because this may cause a kernel heap problem.

4.1.3.7 mallocsize setting

The mallocsize parameter specifies the size of the mallocpool buffer. Like the pagepool buffer, the mallocpool buffer is pinned memory and, therefore, care needs to be taken with setting it to an appropriate value. The mallocpool buffer can vary from 2 MB to 128 MB in size with the default setting being 4 MB. Because of the sensitivity of this parameter, we recommend its setting is considered individually for each GPFS application node.

The current setting of mallocsize on an individual node can be determined by inspecting the file, /var/mmfs/etc/mmfs.cfg. For example:

```
[v06n09:/var/mmfs]# grep "^mallocsize" /var/mmfs/etc/mmfs.cfg
mallocsize 4M
[v06n09:/var/mmfs]#
```

The correct setting of this parameter is closely related to the maxFilesToCache parameter. Setting the parameter too low could seriously degrade the performance of applications that use and/or re-use a large number of files. It might also cause ENOMEM type read errors when attempting to access a large number of files from the file system. On the other hand, setting it too high could seriously impact systems performance and also may not have any beneficial affect where applications are not making use of a large number of files.

The following rule of thumb can be used for setting a value for mallocsize:

mallocsize = 2 * (maximum I-node size for the file system + 800) *maxFilesToCache

On memory constrained systems, it may not be realistic to set this parameter to the value stipulated by the rule. In this case, a balanced decision must be made as to the maximum setting that is not going to reduce pageable virtual memory below an acceptable value.

This parameter can be modified with the `mmchconfig` command. For example:

```
# mmchconfig mallocsize=8M v06n09
```

This would change the mallocsize parameter to 8 Mb on node v06n09. Please note that this change is not dynamic. The GPFS file system must be unmounted, the mmfs daemon restarted and the file system remounted on the target nodes for the change to take effect. Nevertheless, for applications that make dedicated use of the GPFS file system, commands to resize the malloc buffer could be wrapped around an application execution script.

The `mmfsadm dump malloc` command may be helpful in determining whether the mallocpool is running out of memory.

4.1.3.8 worker2Threads setting

This parameter is considered medium impact because it controls the number of threads used by the mmfs daemon to exercise control over directory access and other miscellaneous I/O operations. It is not as high impact as worker1Threads but may be worthy of attention for applications that make use of large numbers of directories or files.

The minimum value for worker2Threads is 1 and the maximum 12. The default is the maximum, 12. The current setting of worker2Threads can be determined by inspecting the file, /var/mmfs/etc/mmfs.cfg. For example:

```
#[v06n01i:/]# grep "^worker2Threads" /var/mmfs/etc/mmfs.cfg
worker2Threads 24
[v06n01i:/]#
```

A guideline is to set worker2Threads according to the prefetchThreads parameter:

worker2Threads = 0.5 * prefetchThreads

Like the worker1Threads parameter, worker2threads is node specific; so, it can be set to different values on different nodes if this is appropriate to the application.

For applications that are performing very large numbers of directory operations, it would be worth increasing the setting over the default to see if there is any tangible improvement in performance. As with prefetchThreads and worker1Threads, care is cautioned not to set the parameter too high.

The worker2Threads parameter cannot be modified dynamically or with the `mmchconfig` command. The mmfs configuration file, mmsdrcfg1, must be retrieved from the SDR and edited to update the parameter(s). The file should then be returned to the SDR, and GPFS should be restarted on the affected nodes. Please refer to 4.1.2.8 on page 115 for more details of the procedure.

4.1.3.9 thewall

This parameter is normally tuned for SP configurations, but it is mentioned here as a medium impact parameter because if it is not set sufficiently high, GPFS performance can be seriously impacted. This is because it would not be possible to allocated a sufficient number of mbufs used for network communications.

The current value of thewall can be displayed with the `no` command. For example:

```
[v06n01i:/etc]# /usr/sbin/no -o thewall
thewall = 65536
[v06n01i:/etc]#
```

We recommend that this parameter is set to 65536, which equates to an upper bound real memory allocation for network buffers to 64 MB.

The parameter can also be set with the `no` command. For example:

```
/usr/sbin/no -o thewall=65536
```

This change can be made dynamically and will survive until the next system reboot. To make the change permanent, the `no` command should be added to the file, /etc/rc.net.

4.1.3.10 Other TCP/IP parameters

There are a collection of TCP/IP parameters not specifically covered in this section that need to be set, not so much for GPFS, but the SP environment in general. These include:

- thewall
- sb_max
- udp_sendspace
- udp_recvspace
- tcp_sendpace

- tcp_recvspace
- rfc1323

These parameters are generic to SP installations and are discussed in detail in the redbook *RS/6000 SP System Performance Tuning*, SG24-5340.

4.1.4 Low impact issues

This section considers tuning issues that can have a minor effect GPFS file system performance. The areas we look at are:

- VSD cache
- queue_depth setting
- The stripe group manager node
- The striping algorithm
- GPFS daemon priority
- The metadata manager node
- File system parameters

4.1.4.1 VSD cache

It is recommended that IBM Virtual Shared Disk caching is turned off. This is because it is considered overall to be better for performance if the memory that would have been pinned for VSD caching to instead be available as system pageable memory.

Whether or not caching is enabled can be ascertained by executing the `lsvsd -l` command on the VSD server. For example

```
[v06n01i:/]# lsvsd -l gpfs1n1
minor  state server lv_major lv_minor vsd-name                    option    size(MB)
 2      ACT    1      39       1      gpfs1n1                     nocache       8672

[v06n01i:/]#
```

4.1.4.2 queue_depth setting

This parameter is only relevant to GPFS file systems configured over RAID 5 arrays. Changes to it should be considered alongside changes to the max_coalesce parameter. This is a low impact parameter because it is not so critical to GPFS performance over RAID 5 arrays as the max_coalesce parameter and, in our tests, varying queue_depth did not have a significant effect. Nevertheless, having a less then optimum value of queue_depth may mean some of the total I/O bandwidth of the RAID array is wasted.

The queue_depth parameter can be inspected via the `lsattr` command or the smitty fastpath `smitty chgssardsk`. For example:

```
[v06n05:/]# lsattr -El hdisk4
pvid            00008655ca50d3bf0000000000000000 Physical volume identifier  False
queue_depth     48                               Queue depth                 True
write_queue_mod 0                                Write queue depth modifier  True
adapter_a       ssa0                             Adapter connection          False
adapter_b       none                             Adapter connection          False
primary_adapter adapter_a                        Primary adapter             True
reserve_lock    yes                              RESERVE device on open      True
connwhere_shad  8681374B12BE4CK                  SSA Connection Location     False
max_coalesce    0x1e0000                         Maximum coalesced operation True
size_in_mb      67683                            Size in Megabytes           False
location                                         Location Label              True
[v06n05:/]#
```

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 2*(N+P) or 3*(N+P) for a N+P array. This maximizes the chance of reading data from each component of the array in parallel. Higher values may improve performance further.

The queue_depth can be modified by use of the `chdev` command or the SMIT fastpath `chgssardsk`. The file system must first be unmounted. The volume group for the GPFS file system must then be varied off and the logical device removed before making the change. For example:

```
# varyoffvg hdisk4
# rmdev -l hdisk4
# chdev  -l hdisk5 -a queue_depth=48
# mkdev -l hdisk4
# varyonvg gpfs4
```

It is recommended the VSD servers are rebooted after making this change.

In our tests with a 4+P RAID-5 array, it has to be said that no discernible difference in read or write performance was noticed when queue depth was varied between 5, 15, and 25.

4.1.4.3 The stripe group manager node

One GPFS node for every GPFS file system has the responsibility for performing Stripe Group manager services. The Stripe Group 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

One or more specific nodes can be nominated as potential stripe group manager by listing them in the file /var/mmfs/etc/cluster.preferences. If the stripe group manager is not defined in this file, any node may be chosen by the GPFS configuration manager when the file system is created. The configuration manger attempts to balance Stripe Group Managers by selecting a node that is not currently a Stripe Group Manager for any other file system.

If you are not sure which node is acting as stripe group manager, this can be ascertained from the `mmfsadm dump` command. For example:

```
# /usr/lpp/mmfs/bin/mmfsadm dump cfgmgr
Cluster Configuration:
Domain ,  6 nodes in this cluster
  ip address    index  admin node  SG's managed   status  failures/panics  SDR node
  129.40.32.89      2           y             1       up        0 /     0         9
  129.40.32.81      4           n             0     down        0 /     0         1
  129.40.32.93      3           n             0       up        0 /     0        13
  129.40.32.83      5           n             0     down        0 /     0         3
  129.40.32.95      7           n             0       up        0 /     0        15
  129.40.32.91      6           n             0       up        0 /     0        11

Cluster configuration manager is 129.40.32.89 (this node)
Assigned stripe group managers:
  "gpfsmd"  SG mgr 129.40.32.89
  Appointed at 928181342.914665022; done recovery=true
Stripe groups managed by this node:
  "gpfsmd" id 812811C9:3752E52C: status recovered, fsck not active
     mgrTakeover noTakeover
     tmBlocked 0, asyncRecoveryInProgress 0, onetimeRecoveryDone 1
     mgrOperationInProgress 0, logFileAssignmentInProgress 0
     FenceDone 0, aclGarbageCollectInProgress 0
     mounts:    4 nodes:  129.40.32.89:2  129.40.32.93:2  129.40.32.95:2  129.40.32.91:2
     panics:    0 nodes
     unfenced:  4 nodes:  129.40.32.89:0  129.40.32.93:0  129.40.32.95:0  129.40.32.91:0
   log group 1, status in use, index 0, replicas 2, user 129.40.32.89
   log group 2, status in use, index -1, replicas 2, user 129.40.32.93
   log group 3, status in use, index -1, replicas 2, user 129.40.32.95
   log group 4, status in use, index -1, replicas 2, user 129.40.32.91
   log group -1, status available, index 7, replicas 1, user (none)
   log group -1, status available, index 9, replicas 1, user (none)
   log group -1, status available, index 10, replicas 1, user (none)
   log group -1, status available, index 11, replicas 1, user (none)
Phoenix Group Names:Gpfs.1, GpfsRec.1; Group State:Active; Quorum:4 Version:(2:2)
```

The Stripe Group Manager can be configured either on a nominated GPFS node or on a VSD server node. If the specific node is particularly heavily CPU bound, you might consider moving the Strip Group Manager to another node. We recommend that the Stripe Group Manager should not be run on the same node as a VSD server even though doing so may not visibly affect the VSD server performance.

The way to change the Stripe Group Manager node is to edit the file /var/mmfs/etc/cluster.preferences on all nodes and specify the new node. Multiple nodes can be listed in this file, but which node is chosen does not depend on its order in the list. After this file has been edited, stop and restart the mmfs daemon on the node currently running the stripe group manager to force it to change to a new node from the list.

4.1.4.4 The striping algorithm

The file system stripe group is the set of physical disks that are used to store data contained in the GPFS file system. Data may be striped across these disks according to three algorithms:

1. roundRobin
2. balancedRandom
3. Random

The roundRobin is the default and generally preferred method as it optimizes performance. There would normally be little merit in altering this option. The roundRobin method is, however, the slowest algorithm when you want to restripe the file system after adding or removing a disk.

The `mmlsfs` command can be used to ascertain which striping algorithm is currently in effect. For example:

```
[sp6n10:/]# mmlsfs /dev/fs1 -s
flag value          description
---- -------------- ------------------------------------------------------
 -s  roundRobin     Stripe method
[sp6n10:/]#
```

The striping algorithm can be modified by use of the `mmchfs` command. For example:

```
# mmchfs -s balancedRandom /dev/fs1
```

All files created on the gpfs file system after this command is executed will have data striped across the file system's disks with the new stripe method. The `mmrestripefs` command can be used to rebalance the file system if necessary, but this command does not necessarily enforce a restripe of all existing data just because the stripe method has changed.

4.1.4.5 GPFS daemon priority

By default, GPFS daemons run at a system priority of 40. It is possible to modify this default although, as a general guideline, we recommend not to. In situations where I/O operations are particularly time critical, consideration could be given to increasing the priority. Caution is recommended when considering changes to the priority of GPFS daemons. Setting them to an inappropriate value could seriously impact system performance. We specifically recommend that the GPFS daemon runs at a higher priority than the VSD daemons and a lower priority than Topology Services (hats). (To ascertain the priority of the VSD daemons, use the command `lssrc -g vsd`

and then `ps -p <pid> -l` on each of the two processes identified by the `lssrc` command.)

In the situation where GPFS applications are running on the same nodes as CPU intensive applications, which are bottle-necking the processor, we recommend reducing the priority of the applications with the `nice` command rather than increasing the priority of the GPFS daemon.

The current GPFS daemon priority on a node can be determined by use of the following command on that node:

```
# grep "^priority" /var/mmfs/etc/mmfs.cfg
```

The GPFS daemon priority can be modified with the `mmchconfig` command. The change will not take affect immediately but will when the daemon is restarted. The following example will reset the daemon priority:

```
# mmchconfig priority=42 v06n01i
# stopsrc -s -c -s mmfs
# startsrc -s mmfs
```

This would change the GPFS daemon priority to 42 on node v06n01i.

4.1.4.6 The metadata manager node

There is one metadata manager for every file open in the GPFS file system. It is not possible to define a specific node to always act as metadata manager for a file system or a specific file. Rather, the GPFS node that first opens a file acts as metadata manager for that file. The node remains the metadata manager until there is no other node with the file open. Where the file is accessed by multiple nodes, and some of these nodes are particularly heavily stressed, there may be some merit in considering opening the file first from a relatively quiet node.

4.1.4.7 File System parameters

There are a couple of file system parameters that are important to set correctly in order to support the required maximum file system size and number of files. These are the file system indirect size and the I-node size. These are both low impact parameters in that they relate more to the file system configuration option than a performance tuning parameter.

A table of options for these parameters is presented in Appendix B of the redbook *GPFS: A Parallel File System*, SG24-5165.

Please note that changing these parameters necessitates a complete rebuild of the file system.

4.2 Tuning verification

Tuning verification in a client-server architecture traditionally comes down to one perspective: Make sure your server performs optimally then add as many clients to the system until client performance is at a minimally acceptable level. This holds true for GPFS, for the most part, but a second more important, but less obvious perspective also applies: Throttle the client's performance so as to not overload the servers. This perspective will be explained in one of the following verification considerations:

- An obvious aspect of verifying how well tuned your system is in relation to GPFS is knowing what your RS/6000 SP system is capable of. As you would expect, hardware product and release differences, system software configuration differences, and application architecture go along way to make similar systems perform worlds apart.

  For each of these areas, it is recommended that you:

  1. First, follow the discussed example systems described in Chapter 3, "Sizing GPFS" on page 71 to gain an insight of what your RS/6000 SP hardware environment is capable of.
  2. In terms of system software configuration for GPFS and related software subsystems, your configuration has probably started with something close to the default settings as recommended in your installation guide.
  3. Finally, your application will constrain your tuning capability to a certain extent, and issues relating to this must be considered as seen in Chapter 2, "Application considerations" on page 57.

- With an understanding of the capability of your system, the next task to perform is a measure of performance and system errors. Measuring tools for both of these are discussed in latter subsections. However, performing these measurement tasks before and after system tuning changes is also recommended. This allows you to verify the improvements or losses in performance or errors that your adjustments have caused.
- Perhaps the most important factor in verifying your systems tuning is not to over-tune your client nodes such that server nodes are over-run with handling client requests. This may seem obvious, but the architecture of GPFS at Version 1.2 and earlier require this. This is because the GPFS architecture does not allow for an elegant way of automatically throttling the client nodes. For this reason, it is possible that enough clients, or reciprocally, not enough servers, could cause file system unavailability problems.

With these points in mind, the following sections will discuss where to look for problem areas and how to measure them. With GPFS, performance problems can occur at any of the layers in the path from the client system application to the servers disk system, as shown in Figure 2 on page 5, and all layers should be considered when trying to verify system tuning.

4.2.1 Monitoring at the server

At the IBM Virtual Shared Disk server, on a dedicated server, tuning problems are not GPFS related. Typically, problems are seen within statistics related to IBM Virtual Shared Disk software; however, monitoring that alone doesn't provide the solution.

The areas that need to be monitored on the server include:

- Switch adapter statistics
- IBM Virtual Shared Disk statistics
- Disk sub-system statistics

4.2.1.1 Analyzing the SP Switch and SP Switch adapter

Switch adapter performance issues on the server can be categorized into one of two problems: The switch fabric is congested, or there are a lack of buffers available in the IBM Virtual Shared Disk architecture. In either case, problems will show up as dropped packets related from the switch adapter. This can be easily seen by using the `netstat` command as shown:

```
[serv6:/]# dsh -av 'netstat -D | grep css' | dshbak
HOST: v06n01i
-------------
css_if0                    3917592             3086871          0          5
HOST: v06n03
------------
css_if0                    3924400             3056409          0         13
HOST: v06n05
------------
css_if0                     395328              237445       1284          0
HOST: v06n07
------------
css_if0                    3932729             3062942        515          6
HOST: v06n09
------------
css_if0                    2250197             5341963          0          2
HOST: v06n11
------------
css_if0                    2150492             4942095          0          2
HOST: v06n13
------------
css_if0                     989943             2779461          0          4
HOST: v06n15
------------
css_if0                    1256402             4336476          0          4
```

The five columns listed in the previous output show device name, `css_if0` for the switch adapter, packets received, packets sent, dropped input packets, and dropped output packets. The last two columns are obviously the important ones as they indicate that nodes five and seven are having trouble handling the incoming packet rate. To determine which problem type this is, further analysis is necessary.

Because the previous screen capture also shows little or no output packet drops, switch congestion is not the issue. If it were, the switch should be analyzed and checked. However, the main errors seen here are input packet drops which indicate a buffer related problem. That is, the server is attempting to send or receive a packet, but either the relevant switch memory pool is full, or there are no mbufs available. A third scenario, the IP interrupt queue, is also a possible place for investigation; however, this will not show up as dropped input packets problem.

- A send pool problem can be checked in a straightforward manner by using either the `vdidl3` or `vdidl3mx` command, depending on your switch adapter. By running it with the `-i` option, as the following output shows, you can check for failures in block allocations.

  Note that this output has been edited to only show send pool statistics.

```
[serv6:/]# dsh -w v06n05,v06n07 'vdidl3 -i' | dshbak
HOST: v06n05
------------
get ifbp info...

send pool: size=16777216 anchor@=0x50018600 start@=0x52550000 tags@=0x524f5000
bkt   allocd     free  success     fail    split     comb    freed
 12        0        0      343        0     1094        0        0
 13        0        0      131        0      131        0        0
 14        1        1     4914        0     7349        0        0
 15        0        1        0        0        0        0        0
 16        4      251    49308        0        0        0        0
HOST: v06n07
------------
get ifbp info...

send pool: size=16777216 anchor@=0x50018600 start@=0x52550000 tags@=0x524fd000
bkt   allocd     free  success     fail    split     comb    freed
 12        0        0      314        0     1075        0        0
 13        0        0      128        0      128        0        0
 14        0        0     4887        0     7397        0        0
 15        0        0        0        0        0        0        0
 16        0      256    49282        0        0        0        0
```

The fail column indicates whether there were any failures in allocating space in the send pool. That is, there isn't enough switch send pool memory to allocate a required buffer size, the split column shows how many times a larger buffer size was split to allocate space for smaller buffer requests. The buffer size can be calculated from the send bkt column as Table 11 shows.

Table 11. Table showing send bucket size to buffer size

Send Bucket Size	Actual Buffer Size (kB)
12	4
13	8
14	16
15	32
16	64

As you can see from the previous example, both nodes five and seven are not having any buffer allocation failures; so, the problem experienced in this example is not send pool related.

Unlike the send pool, the receive pool is checked through the AIX error logs by running the command:

```
errpt -a | grep ENOBUF
```

An error report of this type indicates that the system is running out of receive pool space. Error reports that show mbuf pool threshold exceeded are indicative of receive pool related problems but, more specifically, are mbuf problems.

- As packets continually require mbufs, you should also perform a `netstat -m` to see if there were any failed allocations for small (256 byte) packets. Adjusting the value of thewall in the `no` options can solve mbuf allocation problems. A typical mbuf problem is shown in the following `netstat` output for node 5.

```
[serv6:/]# dsh -w v06n05 'netstat -m' | dshbak
HOST: v06n05
------------

Kernel malloc statistics:

******* CPU 0 *******
By size       inuse     calls failed    free   hiwat   freed
32              114      7162      0      14     640       0
64               79      5030      0      49     320       0
128              40     16379      0      24     160       0
256             326   1178266    972     362     384       3
512              50      9093      0      14      40       0
1024             10      2316      0      18     100       0
2048              0       935      9     100     100     142
4096              1       584      0     104     120      22
16384             0      1309      0      24      24      32
32768             1         1      0       0    2047       0
```

- In a similar fashion to mbuf allocation, the IP interrupt queue also needs monitoring for overruns. This is important as a problem here can not only affect GPFS performance but also other IP using applications.

  The IP interrupt queue can be checked with a `netstat -p IP | grep ipintrq` as the following example shows.

```
[serv6:/]# dsh -a netstat -p IP | grep ipintrq
v06n01i:        0 ipintrq overflows
v06n03:         0 ipintrq overflows
v06n05:         0 ipintrq overflows
v06n07:         0 ipintrq overflows
v06n09:         0 ipintrq overflows
v06n11:         0 ipintrq overflows
v06n13:         0 ipintrq overflows
v06n15:         0 ipintrq overflows
```

As you can see from the previous output, this situation did not suffer from IP overruns. Had there been any problems, it would be necessary to examine the value for `ipqmaxlen`.

Once the problem is located, it is then a matter of resolving it. Typically, if the server to client ratio is sized such that the servers are not being overloaded by the clients, then it is likely a switch tuning parameter on the server has been mis-tuned. For information on tuning areas relating to the SP Switch adapter, refer to Table 12.

Table 12. Tuning verification parameters for the Switch adapter.

Symptom	Relevant Tuning Parameter or Impact Area	Reference
Congested switch fabric	N/A	*Understanding and Using the SP Switch,* SG24-5161
Send pool failures	spoolsize	4.1.2.7, “Switch receive pool and send pool settings” on page 114
Receive pool failures	rpoolsize	4.1.2.7, “Switch receive pool and send pool settings” on page 114
Memory buffer failures	thewall	4.1.3.9, “thewall” on page 127
IP Interrupt queue overrun	ipqmaxlen	4.1.3.4, “ipqmaxlen setting” on page 123

4.2.1.2 Analyzing IBM Virtual Shared Disks

As noted earlier, IBM Virtual Shared Disk server statistics usually indicate problems with the server especially for an overloaded system with a lot of write activity. Ironically, this is easily detected on GPFS nodes by checking for retries and rejected responses. This is performed using the `statvsd` command as demonstrated:

```
[serv6:/]# dsh -N gpfsnodes 'statvsd | egrep "retries|rejected responses"' | dshbak
HOST: v06n09
------------
       237 rejected responses
retries: 1922 859 297 106 33 8 2 0 0
      3227 total retries

HOST: v06n11
------------
       264 rejected responses
retries: 2162 869 277 101 36 13 3 0 0
      3461 total retries

HOST: v06n13
------------
       238 rejected responses
retries: 2028 865 303 113 52 13 1 0 0
      3375 total retries

HOST: v06n15
------------
       230 rejected responses
retries: 2289 858 235 76 17 3 0 0 0
      3478 total retries
```

The retry values are indications of how many requests failed to be serviced by a virtual shared disk server. The first number in the retries list is the number of failures after waiting two seconds. Thereafter, the client waits twice as long as the previous interval until it succeeds. Similarly, the rejected responses indicate that a server sent a response back to the client after it had performed a retry for the packet as shown in the previous screen capture.

Reciprocally, servers collect errors, such as rejected requests, which happen when a server receives a request that has already been processed and rejected merge time-outs, which occur when the server has waited too long for a packet to arrive for a data block to be written. This is shown in the following screen capture:

```
[serv6:/]# dsh -N vsdservers 'statvsd | grep "rejected"' | dshbak
HOST: v06n01i
-------------
         6 rejected requests
         0 rejected responses
         0 rejected no buddy buffer
         4 rejected merge timeout.

HOST: v06n03
------------
        12 rejected requests
         0 rejected responses
         0 rejected no buddy buffer
         8 rejected merge timeout.

HOST: v06n05
------------
       553 rejected requests
         0 rejected responses
         0 rejected no buddy buffer
      1701 rejected merge timeout.

HOST: v06n07
------------
         9 rejected requests
         0 rejected responses
         0 rejected no buddy buffer
         5 rejected merge timeout.
```

The two previous screen captures, therefore, show a typical example of a server, v06n05 in this case, being overrun. A lot of retries on the GPFS nodes and many rejected requests/merge time-outs on the server. This is either because the GPFS node to IBM Virtual Shared Disk server ratio is too high and, therefore, the servers are kept too busy, or the server is mis-tuned and is not performing well.

But what caused the rejections from the server? By examining further statistics from `statvsd`, we can see from the following example that buddy buffers were high on demand, and occasionally pbufs, also known as read/write request, were also unavailable when needed. Request blocks queuing statistics are also list here; however, in this case, buddy buffer shortage is by under much higher demand than any other buffer.

```
[serv6:/]# dsh -w v06n05 'statvsd | grep "queue"' | dshbak
HOST: v06n05
------------
         0 requests queued waiting for a request block
        12 requests queued waiting for a pbuf
         0 requests queued waiting for a cache block
    589345 requests queued waiting for a buddy buffer
       0.0 average buddy buffer wait_queue size
```

If the buffers indicated previously are configured as suggested in 4.1.2.3, “Buddy buffer considerations” on page 105 and 4.1.2.4, “pbuf considerations” on page 107, then following the flow of control through GPFS and data through internal buffers as shown in Chapter 1, “GPFS architecture” on page 1 would lead you to see that a write request at that point can only be slowed by AIX’s Logical Volume Management or related server hardware, such as the IBM Virtual Shared Disk server’s bus, SSA adapter, or disk performance.

Like the SP Switch adapter analysis in “Analyzing the SP Switch and SP Switch adapter” on page 135, Table 13 lists the various impact areas that should be investigated to verify correct tuning of the IBM Virtual Shared Disk server.

Table 13. Tuning verification parameters for the IBM Virtual Shared Disk server

Symptom	Relevant Tuning Parameter/Impact Area	Reference
Retries (client)/ Rejections (server)/ Buddy buffer queued/ Pbuf queued/ Request block queued	buddy buffers	4.1.2.3, “Buddy buffer considerations” on page 105
	pbuf	4.1.2.4, “pbuf considerations” on page 107
	request block	4.1.2.5, “Request block limit” on page 110
	Disk Subsystem	

4.2.1.3 Analyzing the disk subsystem

In general, disk activity problems will fall into three categories: All disks are being utilized at close to one hundred percent of the time, and the disks are transferring as best they can; all disks are still highly utilized, but performance is a little down on their rating; or one disk or all disks on a loop are still extremely busy, but read and write performance is very poor, perhaps as low as fifty percent of normal throughput.

1. The first situation is a typical instance where there simply are not enough physical disks being used in the file system to provide enough throughput. Typically, the command `iostat` will display something like the following output if this scenario occurs:

```
[v06n05:/]# lsvg
rootvg
gpfs15
gpfs10
[v06n05:/]# lsvg -p gpfs10 gpfs15
gpfs10:
PV_NAME           PV STATE    TOTAL PPs   FREE PPs    FREE DISTRIBUTION
hdisk10           active      537         0           00..00..00..00..00
hdisk40           active      537         0           00..00..00..00..00
gpfs15:
PV_NAME           PV STATE    TOTAL PPs   FREE PPs    FREE DISTRIBUTION
hdisk15           active      537         0           00..00..00..00..00
hdisk48           active      537         0           00..00..00..00..00
[sp6n01:/]# iostat -d hdisk10 hdisk40 hdisk15 hdisk48 5
Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk10         99.6     5019.6      41.6          0     25098
hdisk15         99.2     4815.0      41.4          0     24075
hdisk40         99.2     5019.6      41.6          0     25098
hdisk48         98.6     4815.0      41.4          0     24075
```

   If this does occur, and it is combined with the IBM Virtual Shared Disk problems as outlined in the previous section, then two solutions are possible: Either reduce the clients ability to overload the server by throttling their IO ability, or increase disk resource performance on the servers through using a higher performance IO subsystem. The throttling of clients can be achieved through the adjustment of worker1 and worker2 threads.

2. The second situation is a harder to define as all users perceive their disk systems to never operate at optimal performance levels. However, the purpose of this point is to draw attention to tuning SSA or verifying that various tuning options are enabled and set to correct values. There are three items that are worth checking:

   1. Queue Depth
   2. Maximum Coalesce
   3. Fast Write Cache (for RAID 5 only)

   There is no way to monitor the direct impact of these SSA disk attributes; however, it is possible to at least verify settings as recommended in earlier sections.

3. In the third situation, as you can see in the next `iostat` output, the disks still have high utilization rates; however, throughput has dropped off significantly.

```
[v06n05:/]# iostat -d hdisk10 hdisk40 hdisk15 hdisk48 5
Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk10         81.2     1851.2      37.4          0      9256
hdisk15         84.0     1849.8      37.4          0      9249
hdisk40         80.2     1851.2      37.4          0      9256
hdisk48         83.4     1849.8      37.4          0      9249
```

   This situation would suggest that either the disk is damaged and is continually attempting internal recovery procedures, or the disk attachment path is very busy. For a damaged disk, the obvious solution is to investigate relevant error reports, and then the failing disk or SSA adapter device should replaced as per normal maintenance procedures.

   For a congested data path problem, the server environment should be redesigned to reduce congestion. Among other things, this could be done by migrating some disks to either a quieter loop or a new loop altogether. For further information regarding sizing of disks per SSA loop, refer to 3.1.3, “Servers” on page 80.

In any of these situations, the disk subsystem performance needs to be tuned or reviewed to address sizing issues. Direct software tuning is possible but limited. This is summarized in Table 14.

Table 14. Tuning verification parameters for the disk sub-system

Symptom	Relevant Tuning Aspect or Parameter	Reference
High utilization, high throughput	Server sizing	Chapter 3, “Sizing GPFS” on page 71
High utilization, moderate throughput	Maximum Coalesce and Queue Depth	4.1.2.10, “max_coalesce setting” on page 118 and 4.1.4.2, “queue_depth setting” on page 128.
High utilization, low throughput		

4.2.2 Monitoring at the client

The GPFS client verification is obviously a lot different from the IBM Virtual Shared Disk servers verification because of the operational requirement that the client has. That is, the client, in terms of performance, is only limited to

how well it can issue I/O requests through GPFS. System latency is always a factor; however, SP Switch congestion may also be a factor. Unlike latency, switch congestion is possible to monitor and tune.

But, this is still answering the question as if there is a bottleneck between the client and server. At this point in the tuning verification, you may still have your clients out performing your IBM Virtual Shared Disk servers. As an alternative to the question of verifying that a particular server setup is tuned, this section will also attempt to address the question: How do you tune a system that already has the servers well tuned but still has the clients overrunning them?

4.2.2.1 SP Switch analysis

There is only one area worth investigating for client node tuning, and that is switch congestion. As with the server switch analysis performed in section 4.2.1.1, “Analyzing the SP Switch and SP Switch adapter” on page 135, the testing is again performed by using the `netstat -D` command. There is not likely to be many input packet drop errors unless the GPFS node is also performing another server like function that causes it to not be able to receive packets.

The following screen capture demonstrates that all of the GPFS nodes are not suffering from too much switch congestion. This is shown by only a few output packets being dropped.

```
[serv6:/]# dsh -N gpfsnodes 'netstat -D | grep css' | dshbak
HOST: v06n09
------------
css_if0                     2250197             5341963          0          2
HOST: v06n11
------------
css_if0                     2150492             4942095          0          2
HOST: v06n13
------------
css_if0                      989943             2779461          0          4
HOST: v06n15
------------
css_if0                     1256402             4336476          0          4
```

Given the proportion of dropped output packets here, congestion is not hindering the performance of the clients.

4.2.2.2 Measuring the client

Emphasis to this point has been that some hardware constraint within the IBM Virtual Shared Disk server, or the SP Switch itself, is the item or items

that hinder the throughput from GPFS clients to IBM Virtual Shared Disk servers. It should be obvious then that an over-sized server environment can lead to the situation where the servers are, in effect, idling. This then leads to the question of how to know how many clients can be added until the server is busy but not overrun?

In an existing environment, this should be a straightforward process:

1. Measure how much throughput your IBM Virtual Shared Disk servers can provide.
2. Measure how much throughput your existing GPFS nodes create.
3. Calculate both the available server throughput and the throughput rate per GPFS node.
4. The amount of times the GPFS node rate can go into the available throughput is approximately the number of clients that can be added.

But what if your GPFS nodes already overrun your servers, and you can not resize your server environment? The alternative is to reduce the throughput capability of the GPFS node. As with GPFS Version 1.2, the mechanism for controlling this is a manual one in adjusting the number of worker and prefetch threads. For information on setting these parameters, refer to 4.1.2.9, “worker1Threads setting” on page 117, 4.1.2.8, “prefetchThreads parameter” on page 115, and 4.1.3.8, “worker2Threads setting” on page 126.

So what is best for your RS/6000 SP? The first step is to gauge what a worker thread is worth for your application. By setting your worker thread values to an unrealistically low value and then increasing the value, you can see performance, and possibly problems, increase and decrease. The following example demonstrates the results of a simple, two IBM Virtual Shared Disk server environment with four GPFS nodes. The application was a single write `dd` command on each GPFS node with increasing worker one threads. Each test was repeated four times to produce an average and standard deviation result. The results are shown in Table 15., “Worker thread performance example” on page 146, and for visual representation, are also shown in Figure 29., “Worker thread performance example” on page 148.

Table 15. Worker thread performance example

Worker Threads per Client	Average Time (secs.)	Standard Deviation	Dropped Packets	Other Information
1	259.29	0.23	0	Disk Utilization was low - approximately 20% on average

Worker Threads per Client	Average Time (secs.)	Standard Deviation	Dropped Packets	Other Information
2	138.94	0.33	0	
4	125.78	0.62	0	
6	125.24	0.81	0	Disk utilization seemed to peek here at around 50-55%, with nearly 100% CPU wait time.
9	126.25	1.28	0	
12	130.01	0.72	0	
15	129.55	1.00	10	Dropped Packets occurred late in the tests, and gradually increased.
20	153.65	5.41	200	Dropped packets started early and came fast. I/O wait time was also reduced noticeably on other servers.

This environment was chosen because it was a simple task to over run the servers with a low number of worker1 threads on the GPFS nodes. The results show that, at approximately six worker1 threads per client, the application maximized the capability of the servers without causing any errors.

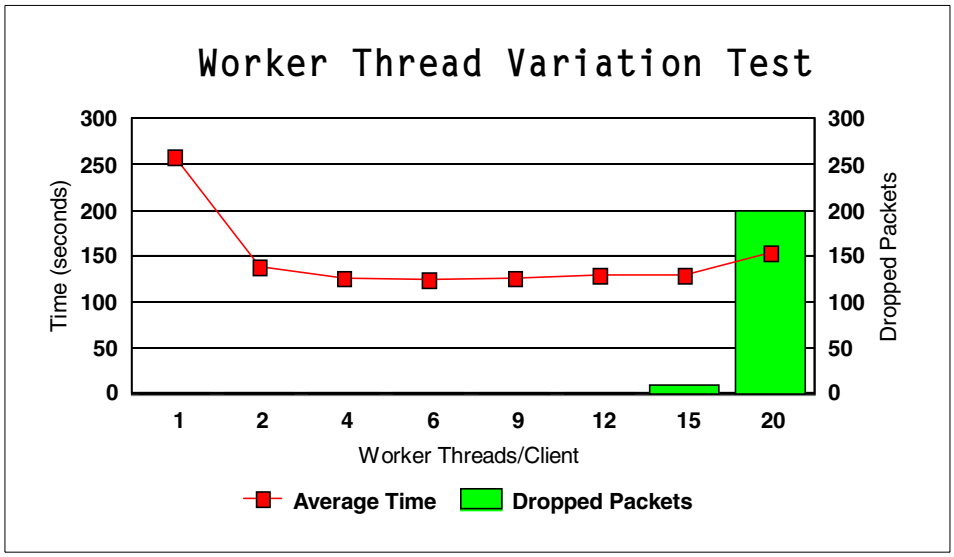


Figure 29. Worker thread performance example

It is also interesting to note that, in this example, there is little difference in the performance between approximately four and nine worker threads per node, that is, a system wide thread total of between sixteen and thirty six threads. This indicates that there is some flexibility in threads per node in a small environment. Should the RS/6000 SP system grow without increasing the collective server performance, the total number of worker threads should be evenly redistributed. For example, to go to an eight GPFS node system, each node should have between two and four worker threads.

Once above this total level, packets begin to be dropped at increasing rates, and then the variation in the run time increases indicating that the servers are now becoming swamped and are no longer behaving deterministically.

With this as a template for measuring GPFS node performance, it should now be possible to apply this simple, although effective, technique to other similar server bound systems.

4.2.3 Interpreting the numbers

Having resolved where the problem symptoms are occurring within the GPFS architecture of your RS/6000 SP system, the final part in verifying your system is to put all the testing together to determine what the problem is. What you have seen so far is that the server is very heavily tuned and tested and is nearly always the constraining factor in the system. But because the

server uses a number of interacting subsystems to achieve its goal of GPFS disk serving, a single symptom in one area may not necessarily indicate the true or only problem with the system. In fact, it is possible with a well tuned server environment that a tuning problem in one area is likely to be the result of a tuning problem in an adjoining subsystem.

In some situations, such as with the IBM Virtual Shared Disk server's disks, the measurements alone of that subsystem can indicate the problem. In others, such as the IBM Virtual Shared Disk subsystem, retries can indicate either a mis-tuned IBM Virtual Shared Disk subsystem, or disk subsystem problems, or both.

A good way to visualize the flow of possible symptoms and subsystem interaction is shown in Figure 30 on page 149, which shows that if symptoms are measured in one of the subsystems discussed earlier, the tuning problem may, in fact, be occurring elsewhere and, in general, be heading back towards the IBM Virtual Shared Disk server's disk subsystem.

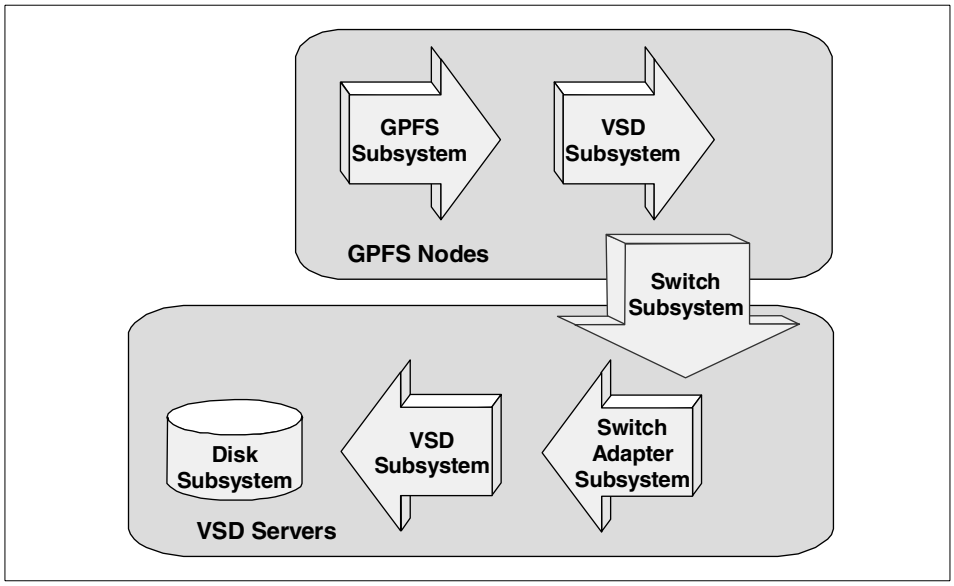


Figure 30. Flow of symptoms within a GPFS architecture

The first area to begin analyzing the system is at the IBM Virtual Shared Disk server. The check for retries as described in section 4.2.1.2, "Analyzing IBM Virtual Shared Disks" on page 139 should be performed because this is the prime indicator for server overruns caused by any of the subsystems. System checks can then be further analyzed by using the following points as a guide:

- If outgoing packets are being dropped by the SP Switch adapter on either the server or the GPFS node, then there is a switch or adapter congestion problem.
- If client requests are being queued at the server, then either the IBM Virtual Shared Disk resources are under configured, or there is a disk subsystem problem.
- Otherwise, incoming packets are being dropped by the switch or the IP interface because of switch pool shortages, mbuf shortages, or IP interrupt queue shortages.

Tuning should then occur in the relevant areas to try to alleviate problems. Hardware problems, such as congestion or errors, should be checked and corrected. If the server is tuned according to the guidelines of this chapter, and server overruns are still occurring, then the two remaining options are:

- Increase the number of servers or performance of servers
- Throttle the clients IO activity

These should be tried until a balance is achieved. Verification procedures should also be repeated regularly to confirm that the GPFS architecture is operating correctly. The following changes should at least provide a quick analysis of the IBM Virtual Shared Disk servers and GPFS nodes to confirm the absence of tuning problems.

- Application IO operation profile, for example, changing from a mainly sequential access application to a random access application,
- RS/6000 SP's hardware, for example, replacement of hard disks with the same volume but faster performance characteristics, and
- RS/6000 SP's system software, for example, applying a set of program temporary fixes (PTFs)

4.3 Tuning case studies

This section describes an experiment we carried out to explore the impact of tuning GPFS parameters on the GPFS performance by running parallel and random I/O-intensive applications.

The objectives of the case study are:

- To provide you with the optimal values for all GPFS parameters so that you can achieve the best results when running any application on GPFS.
- To extract as much useful information as possible from the case study.

4.3.1 Hardware, software, and GPFS configuration

This section lists the hardware and software configuration used for the test case study.

Hardware configuration

1. One SP Frame
2. Two VSD server nodes:
    - Four POWER2 processor (332 MHz) each, PCI bus
    - 256 MB RAM
    - One SSA adapter for each server
    - One loop of five 4.5 SSA disk on each server node
    - CSS switch network
3. Five thin GPFS nodes
    - MCA BUS nodes
    - One POWER processor (62 MHz) each
    - 256 MB RAM
    - CSS switch network

Software configuration

Each node is loaded with the following software:

- AIX 4.3.2.0 with Jan,1999 PTF set
- PSSP 3.1.0.4
- RVSD 3.1.0.3
- GPFS 1.2.0.2

GPFS configuration

The GPFS configuration has two VSD servers configured as a dedicated VSDs, with one SSA loop for each VSD server, having five 4.5 GB disks in each loop. Figure 31 on page 152 shows the hardware configuration of GPFS.

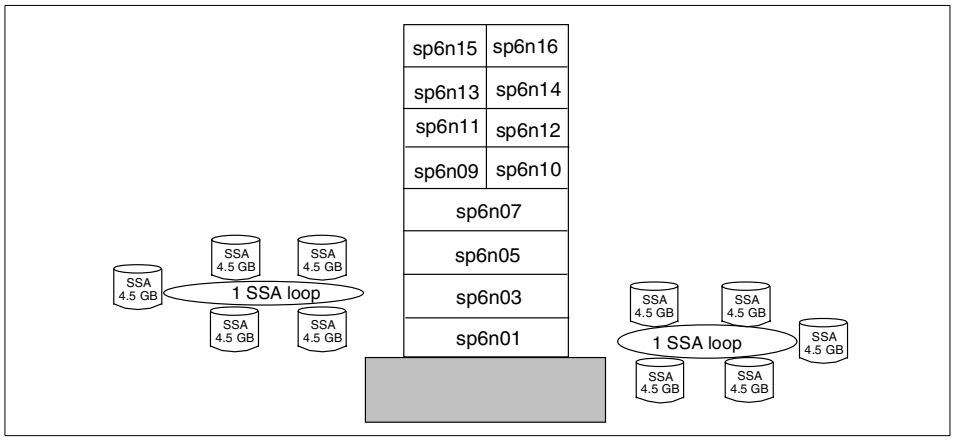


Figure 31. Hardware configuration for GPFS

The initial tunable parameters and buffers of GPFS and SP are configured as shown Table 16.

Table 16. Initial parameter of the system

Parameter	Value
Switch rpoolsize and rpoolsize	16 MB
thewall	64 MB
ipqmaxlen	512
Buddy buffer on VSD servers	33
Buddy buffer on GPFS nodes	2
pagepoolsize on GPFS nodes	20
pbuf	48
max_IP_msg_size	60 KB
maxFilesToCache on GPFS nodes	200
mallocsize on GPFS nodes	4 M
prefetchThreads	48
worker1Threads	72
worker2Threads	24
priority	40

The Virutal Shared Disk (VSD) parameters for each node are entered into the SDR as shown in the following screens.

```
sp6en0>> /usr/lpp/csd/bin/vsdatalst -n

VSD Node Information
                                    Initial  Maximum  VSD      rw      Buddy  Buffer
node                    VSD   IP packet  cache    cache   request request minimum maximum size: #
number host_name      adapter    size    buffers  buffers  count   count   size    size   maxbufs
------ ---------------- ------- ------- -------- -------- ------- ------- ------- -------
1  sp6n01.msc.itso css0  61440    64      256     256     48    4096   262144 33
3  sp6n03.msc.itso css0  61440    64      256     256     48    4096   262144 33
10 sp6n10.msc.itso css0  61440    64      256     256     48    4096   262144 2
11 sp6n11.msc.itso css0  61440    64      256     256     48    4096   262144 2
12 sp6n12.msc.itso css0  61440    64      256     256     48    4096   262144 2
13 sp6n13.msc.itso css0  61440    64      256     256     48    4096   262144 2
14 sp6n14.msc.itso css0  61440    64      256     256     48    4096   262144 2
```

```
sp6n10>> /usr/lpp/csd/bin/lsvsd -l

minor  state server lv_major lv_minor vsd-name     option    size(MB)
  2      ACT    1       0        0      gpfs1n1     nocache    4296
  3      ACT    1       0        0      gpfs2n1     nocache    4296
  4      ACT    1       0        0      gpfs3n1     nocache    4296
  5      ACT    1       0        0      gpfs4n1     nocache    4296
  6      ACT    1       0        0      gpfs5n1     nocache    4296
  7      ACT    3       0        0      gpfs6n3     nocache    4296
  8      ACT    3       0        0      gpfs7n3     nocache    4296
  9      ACT    3       0        0      gpfs8n3     nocache    4296
 10      ACT    3       0        0      gpfs9n3     nocache    4296
 11      ACT    3       0        0      gpfs10n3    nocache    4296
```

The GPFS file system is created with the following stanza and configuration as shown in the following screen.

```
/gpfs:
                    dev                                     = /dev/fs1
                    vfs                                     = mmfs
                    nodename                                = -
                    mount                                   = mmfs
                    type                                    = mmfs
                    options                                 =
rw,disks=gpfs1n1;gpfs6n3;gpfs2n1;gpfs7n3;gpfs3n1;gpfs8n3;gpfs4n1;gpfs9n3;gpfs5n1;gpfs10n3
                    account                                 = false
```

The following screens show a snapshot of the GPFS file system configuration.

```
sp6n10>> /usr/lpp/mmfs/bin/mmdf fs1

free KB   free KB
disk            disk size  failure holds    holds    in full        in
name                in KB    group metadata data      blocks fragments
--------------- --------- -------- -------- ------ --------- ---------
gpfs1n1           4399104     4001 yes      yes      4393216       592
gpfs2n1           4399104     4001 yes      yes      4392960       592
gpfs3n1           4399104     4001 yes      yes      4393216       584
gpfs4n1           4399104     4001 yes      yes      4393216       584
gpfs5n1           4399104     4001 yes      yes      4393216       592
gpfs6n3           4399104     4003 yes      yes      4393216       584
gpfs7n3           4399104     4003 yes      yes      4392960       584
gpfs8n3           4399104     4003 yes      yes      4393216       592
gpfs9n3           4399104     4003 yes      yes      4393216       592
gpfs10n3          4399104     4003 yes      yes      4393216       576
                 ---------                         --------- ---------
(total)          43991040                          43931648      5872
nInodes: 46080
free inodes: 46054
```

```
sp6n10>> /usr/lpp/mmfs/bin/mmlsfs fs1

flag value         description
---- ------------- ------------------------------------------------------
 -s  roundRobin    Stripe method
 -f  8192          Minimum fragment size in bytes
 -i  512           Inode size in bytes
 -I  16384         Indirect block size in bytes
 -m  1             Default number of metadata replicas
 -M  1             Maximum number of metadata replicas
 -r  1             Default number of data replicas
 -R  1             Maximum number of data replicas
 -a  1048576       Estimated average file size
 -n  8             Estimated number of nodes that will mount file system
 -B  262144        Block size
 -Q  none          Quotas enforced
 -F  46080         Maximum number of inodes
 -V  2             File system version. Highest supported version: 2
 -d  gpfs1n1;gpfs2n1;gpfs3n1;gpfs4n1;gpfs5n1;gpfs6n3;gpfs7n3;gpfs8n3;gpfs9n3;gpfs10n3  Disks in file system
 -C  1             Configuration identifier
```

```
sp6n10>> /usr/lpp/mmfs/bin/mmlsdisk fs1

disk         driver   sector failure holds    holds
name         type       size   group metadata data  status availability
------------ -------- ------ ------- -------- ----- ------------- ------------
gpfs1n1      disk        512    4001 yes      yes   ready up
gpfs2n1      disk        512    4001 yes      yes   ready up
gpfs3n1      disk        512    4001 yes      yes   ready up
gpfs4n1      disk        512    4001 yes      yes   ready up
gpfs5n1      disk        512    4001 yes      yes   ready up
gpfs6n3      disk        512    4003 yes      yes   ready up
gpfs7n3      disk        512    4003 yes      yes   ready up
gpfs8n3      disk        512    4003 yes      yes   ready up
gpfs9n3      disk        512    4003 yes      yes   ready up
gpfs10n3     disk        512    4003 yes      yes   ready up
```

The above screens show the basic GPFS configuration on our SP system without performing any tuning on any side. This configuration will be used as a base configuration for the following scenarios.

The following sections demonstrate the performance impact in tunning GPFS parameters for different scenarios.

4.3.1.1 Running a parallel test on a JBOD GPFS

In this scenario, we over run the VSD servers by adding more GPFS nodes, and then we tune the system to a balance state.

Initially, GPFS is configured as shown earlier, and we ran the parallel application on five GPFS nodes. The parallel tests were comprised of writing and reading 8 GB files to the GPFS file system. This was done from five client nodes simultaneously. The application block size was always 256 KB, and the data partitioning file layout was always segmented. All tests utilized the ior program.

After running the program, the program aborted complaining about Device /gpfs not available. By looking at the following screen capture, we noticed that the last two columns indicate that VSD server sp6n01 is having trouble handling the incoming packet rate through the switch. To determine which problem type it is, we did further analysis by looking at the `statvsd` command output.

```
[sp6en0:/]# dsh -N vsdservers 'netstat -D | grep css' | dshbak
HOST: sp6n01
-------------
css_if0                     1557788             1814354         56          0
HOST: sp6n03
------------
css_if0                     1520250             1701828         0           4
```

Because the previous screen capture also shows little or no output packet drops, switch congestion is not the issue. However, the main errors seen are input packet drops, which indicate a buffer related problem. That is, the server is attempting to send or receive a packet, but either the switch memory pool is full, or there are not enough mbufs available.

In most cases, when the buddy buffer on the VSD server is full, the switch buffer keeps the packets, and this will fill up the switch buffers.

This might happen if the GPFS node to VSD server ratio is too high and, therefore, the servers are kept too busy, or the VSD server is mis-tuned.

By examining `statvsd` output, we can see from the following screen that buddy buffers are high on demand and, occasionally, pbuf buffers are unavailable when they are needed. Request blocks queuing statistics are also list here; however, in this case, buddy buffer shortage is in much higher demand than any other buffer.

```
[sp6en0:/]# dsh -w sp6n01 'statvsd | grep "queue"' | dshbak
HOST: sp6en0
------------
         0 requests queued waiting for a request block
      2908 requests queued waiting for a pbuf
         0 requests queued waiting for a cache block
    168277 requests queued waiting for a buddy buffer
       0.0 average buddy buffer wait_queue size.
```

To solve this problem, we increase the maximum number of buddy buffers on the VSD servers to 66 by issuing the following command:

```
sp6en0>> updatevsdnode -n 1,3 -s 66
```

Then we restart the `rvsd` and `mmfs` daemons on the cluster machine.

By increasing the buddy buffers on the VSD servers, we allow the VSD server to queue more requests and, therefore, are able to free up the switch rpool buffer.

We ran the program again, and it is completed successfully. Both VSD servers were giving 46 MB/S write and 43 MB/S read, which seem to be reasonable values for the configuration we have.

Also, the switch on the VSD servers did not show any dropped In/Out packets.

Since we are configuring five disks in each loop per server, which means each server will give around 20 to 25 MB/S. This means a total of two servers will give 45 to 49 MB/S, which we accomplished through this test. See Table 17 on page 163 for the final results.

4.3.1.2 Running a random access test on a JBOD GPFS

In this scenario, we run a random application on a non-tuned system, and then we tune the GPFS to provide us with better results.

Initially, the GPFS is configured as shown previously, and we run the random application on five GPFS nodes. The random test was comprised of writing and reading a 256 MB file to the GPFS file system. This was done from five clients nodes simultaneously. The application block size varies between 400 and 40,000 bytes. The offset into the file varies as well. The test utilizes the ior program.

By running the program, we get the following results for read and write I/O operations:

- 0.77 MB/sec for Write operation
- 2.27 MB/sec for Read operation

Most likely, the I/O performance in this situation is not related to the VSD servers because the VSD servers are not highly utilized, which is indicated by looking at the `iostat` and `netstat` output. However, its more related to GPFS and the application program. The following steps are done to improve the I/O performance:

1. Reduce the blocksize of the GPFS file system

   We know, for a fact, that the program write's block size varies between 400 and 40,000 bytes, and our GPFS is configured for 256 KB blocks. In this case, we reconfigure the GPFS file system with 64 KB blocks and run the program again. We get the following results:

   - 0.96 MB/sec for Write operation
   - 2.86 MB/sec for Read operation

   This shows an improvement in the performance of the I/O bandwidth by 25 percent of write and 26 percent of read. More tunning can be done by changing the pagepool buffer size. The next step describes the pagepool sizing.

2. Increase the pagepool size in the GPFS nodes

   As it is mentioned in Section 4.2.2, the pagepool buffer is critical for applications that do random write/read; therefore, we increase the pagepool buffer from 20 MB the default to 45MB. The following command is used to increase the pagepool buffer:

   ```
   /usr/lpp/mmfs/bin/mmchconfig pagepool=45M -i
   ```

   After increasing the pagepool buffer, we run the application, and we get the following results:

   - 1.033 MB/sec for write I/O.
   - 2.97 MB/sec for read I/O.

   As can be seen, the I/O write operation is improved by 34 percent and the read is improved by 30 percent from the default results.

4.3.1.3 Running a parallel test on a replicated GPFS

The GPFS file system is configured as mentioned earlier, except it is replicated. The application used is the same application that has been used for the earlier tests.

We run the parallel application on five GPFS nodes with a 256 KB block size. The data partitioning file layout is always segmented.

By running the program, we get the following results for read and write I/O operations:

- 20.71 MB/sec Write operation
- 43.49 MB/sec Read Operation

By looking at the output of `statvsd`, we find that there are number of requests queued waiting for a pbuf and a number of requests queued waiting for a buddy buffer at both VSD servers. The following screen shows the output of the `statvsd` command.

```
[sp6en0:/]# dsh -w sp6n01 'statvsd | grep "queue"' | dshbak
HOST: sp6n01
------------
         0 requests queued waiting for a request block
     19533 requests queued waiting for a pbuf
         0 requests queued waiting for a cache block
     38876 requests queued waiting for a buddy buffer
       0.0 average buddy buffer wait_queue size

[sp6en0:/]# dsh -w sp6n03 'statvsd | grep "queue"' | dshbak
HOST: sp6n03
------------
         0 requests queued waiting for a request block
      9803 requests queued waiting for a pbuf
         0 requests queued waiting for a cache block
      9611 requests queued waiting for a buddy buffer
       0.0 average buddy buffer wait_queue size
```

The above screen shows that the buddy buffer is high on demand, and the pbuf buffers, also known as read/write request buffers, are also unavailable when needed. In this case, the buddy buffer is in much higher demand than any other buffer.

To solve this problem, we use the following steps:

1. Increased the pbuf and the buddy buffer as recommended in Section 4.2.1.

   The pbuf is set to 80 for each server. Also the buddy buffer is increased to 66 per VSD server as shown in the following command:

   - `sp6en0>> updatevsdnode -n 1,3 -s 66`
   - `sp6en0>> updatevsdnode -n 1,3 -p 80`

The following screen shows the output of the `vsdatalst` after changing the pubuf and the buddy buffer.

```
sp6en0>> /usr/lpp/csd/bin/vsdatalst -n

VSD Node Information
                                        Initial  Maximum  VSD      rw      Buddy  Buffer
node                      VSD   IP packet  cache    cache  request request minimum maximum size: #
number host_name      adapter    size  buffers  buffers  count   count    size   size   maxbufs
------ ---------------- ------- ------- -------- -------- ------- ------- ------- -------
1  sp6n01.msc.itso css0  61440    64      256      256     80      4096   262144 66
3  sp6n03.msc.itso css0  61440    64      256      256     80      4096   262144 66
10 sp6n10.msc.itso css0  61440    64      256      256     48      4096   262144 2
11 sp6n11.msc.itso css0  61440    64      256      256     48      4096   262144 2
12 sp6n12.msc.itso css0  61440    64      256      256     48      4096   262144 2
13 sp6n13.msc.itso css0  61440    64      256      256     48      4096   262144 2
14 sp6n14.msc.itso css0  61440    64      256      256     48      4096   262144 2
```

We run the application and monitor the output of the `statvsd`. The output of the `statvsd` command still shows an increase in the requests waiting for pbuf and buddy buffers on both servers.

```
[sp6en0:/]# dsh -w sp6n01 'statvsd | grep "queue"' | dshbak
HOST: sp6n01
------------
         0 requests queued waiting for a request block
     36091 requests queued waiting for a pbuf
         0 requests queued waiting for a cache block
     28410 requests queued waiting for a buddy buffer
       0.0 average buddy buffer wait_queue size

[sp6en0:/]# dsh -w sp6n03 'statvsd | grep "queue"' | dshbak
HOST: sp6n03
------------
         0 requests queued waiting for a request block
     18510 requests queued waiting for a pbuf
         0 requests queued waiting for a cache block
     23305 requests queued waiting for a buddy buffer
       0.0 average buddy buffer wait_queue size
```

The application completed successfully with increase in the requests queuing for the pbuf and buddy buffer. Both VSD servers were giving almost 20 MB/S write and 43 MB/S read, which seems to be reasonable values for replicated file systems.

Further investigation needs to be done to pin point the problem. While the application was running we took a snapshot of `iostat` on one of the VSD servers as shown in the following screen:

```
tty:      tin     tout  avg-cpu: % user  % sys  % idle  % iowait
          0.0    309.8              0.4   11.4     0.0    88.3

Disks:     % tm_act     Kbps      tps  Kb_read   Kb_wrtn
hdisk0        2.7      15.0       3.7         0        60
hdisk1        0.0       0.0       0.0         0         0
hdisk2        0.0       0.0       0.0         0         0
hdisk3        0.0       0.0       0.0         0         0
hdisk4       99.9    4541.2      38.0         0     18176
hdisk5        0.0       0.0       0.0         0         0
hdisk6        0.0       0.0       0.0         0         0
hdisk7        0.0       0.0       0.0         0         0
hdisk8        0.0       0.0       0.0         0         0
hdisk9        0.0       0.0       0.0         0         0
hdisk10      99.9    4349.3      32.2         0     17408
hdisk11       0.0       0.0       0.0         0         0
hdisk12       0.0       0.0       0.0         0         0
hdisk13      99.9    4614.6      38.2         0     18470
hdisk14      99.9    4606.4      38.5         0     18437
hdisk15      93.9    4736.3      37.5         0     18957
.
```

The above screen shows that the bottleneck are the disks. As we can see, the disks are almost 100 percent utilized and that the requests queued are waiting for the buddy buffer and pbuf.

For further solutions, we perform the following:

- Reduce the worker1Threads and the prefetchThreads at GPFS nodes

  In this situation, we notice that the I/O sub-system is severely over-streached by write I/O requests. I/O requests may stall with a potential for the file system to go off line. To solve this problem, we lower the worker1Threads and the prefetchThreads parameters to bring the write I/O levels back down to a manageable level.

  The worker1Threads and prefetchThreads are set to 9 and 6, respectively. See Section 4.3.2.8 page 92 for more details on how to set those values.

  By running the application and monitoring the `statvsd` command output, we can see that there is no increase in the number of request queues waiting for buddy buffer or pbufs as before. The following screen shows the output of the `statvsd` command on both VSD servers.

```
[sp6en0:/]# dsh -w sp6n01 'statvsd | grep "queue"' | dshbak
HOST: sp6n01
------------
         0 requests queued waiting for a request block
     36171 requests queued waiting for a pbuf
         0 requests queued waiting for a cache block
     28410 requests queued waiting for a buddy buffer
       0.0 average buddy buffer wait_queue size

[sp6en0:/]# dsh -w sp6n03 'statvsd | grep "queue"' | dshbak
HOST: sp6n03
------------
         0 requests queued waiting for a request block
     18610 requests queued waiting for a pbuf
         0 requests queued waiting for a cache block
     23305 requests queued waiting for a buddy buffer
       0.0 average buddy buffer wait_queue size
```

After running the program, we get the following results for read and write I/O operations:

- 20.71 MB/sec Write operation
- 42.71 MB/sec Read Operation

At this stage, the system is in stable state. No requests queued are waiting for pbufs or buddy buffers. Further solutions can be done by increasing the number of disks attached to each VSD server.

4.3.1.4 Running a random test on a replicated GPFS

In this scenario, we run a random application on a non-tuned system, and then we will tune the GPFS to provide us with better results.

Initially, the GPFS is configured, as shown earlier, with replicating data and metadata, and we run the random application on five GPFS nodes. The random test is comprised of writing and reading a 256 MB file to the GPFS file system. This was done from five clients nodes simultaneously. The application block size varied between 400 and 40,000 bytes. The offset into the file was varied as well. The test utilized the ior program.

When we ran the program, we got the following initial results for read and write I/O operations:

- 0.46 MB/sec Write operation
- 2.30 MB/sec Read Operation

Most likely, the I/O performance in this situation is not related to the VSD servers because the VSD servers are not highly utilized, which is indicated by

looking at the `iostat` and `netstat` output. However, its more related to GPFS and the application program. The following steps were taken to improve the I/O performance:

1. Increase the pagepool size in the GPFS nodes

   As it is mentioned in Section 4.2.2, the pagepool buffer is critical for applications that do random write/read. We increased the pagepool buffer from 20 MB, the default, to 45 MB. The following command is used to increase the pagepool buffer:

   ```
   /usr/lpp/mmfs/bin/mmchconfig pagepool=45M -i
   ```

   After increasing the pagepool buffer, we ran the application and got the following results:

   - 0.47 MB/sec for write I/O.
   - 2.35 MB/sec for read I/O.

   As can be seen, increasing the pagepool on the GPFS nodes did not have a big affect in the GPFS performance for this situation. The only improvement we get is 0.02 percent on write and read operations.

2. Reduce the blocksize of the GPFS file system

   We know, for a fact, that the program writes block size varied between 400 and 40,000 bytes and that our GPFS is configured for 256 KB blocks. In this case, we reconfigure the GPFS file system with 64KB blocks and run the program. We get the following results:

   - 0.68 MB/sec for Write operation
   - 3.01 MB/sec for Read operation

   This shows an improvement in the performance of the I/O bandwidth by 50 percent on write and 26 percent on read operation.

Table 17 shows a summary of the test that have been done and the results that we got before and after the tuning.

Table 17. Tuning Results

No	Test Desc	Write/Read I/O Before Tuning	Write/Read I/O After Tuning	% Improv.
1	Parallel Access file with JBOD	Failed: Device /gpfs not ready	46.36 MB/s Write 43.17 MB/s Read	Fixed Problem
2	Random Access file with JBOD	0.77 MB/s Write 2.27 MB/s Read	1.03 MB/s Write 2.97 MB/s Read	34% Write 30% Read
3	Parallel Access files with Replica	20.71 MB/s Write 43.49 MB/s Read	20.71 MB/s Write 42.71 MB/s Read	Fixed Problem

No	Test Desc	Write/Read I/O Before Tuning	Write/Read I/O After Tuning	% Improv.
4	Random Access files with Replica	0.46 MB/s Write 2.38 MB/s Read	0.69 MB/s Write 3.01 MB/s Read	50% Write 26% Read

4.3.1.5 Serial application with a variable block size on GPFS

In this scenario, we run a serial application on a SP system to measure the write and read I/O behavior on a variable block size.

The GPFS is configured as mentioned earlier, and we run the serial application on one GPFS node. The serial tests were comprised of writing and reading an 8 GB file to the GPFS file system. The application block size is varying between 4 K to 4 MB each time. All tests utilize the iopm program.

Figure 32 on page 164 shows that the performance improves until a 256 KB block size and then remains fairly constant above 256 KB. This is to be expected since the GPFS file system is configured with a 256 KB block size.

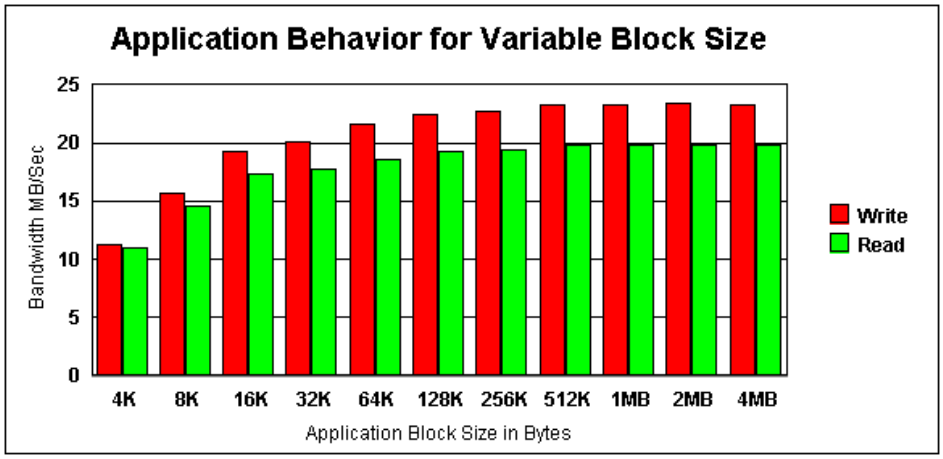


Figure 32. Bandwidth versus block size

Chapter 5. Implementing and tuning ADSM for GPFS

> **Note:**
>
> Tivoli has announced Tivoli Storage Management, formerly IBMs ADSTAR Distributed Storage Manager (ADSM). For the purpose of this redbook, we refer to the product as ADSM. For more information about Tivoli Storage Management, please visit the Tivoli ADSM Website at:
> `http://www.tivoli.com/storage`

This chapter deals with implementation and tuning considerations for back up and restore of GPFS file systems using ADSTAR Distributed Storage Manager (ADSM). The ADSM Version 3.1 backup-archive client on AIX Version 4.2.1 and higher provides backup and archive functions for supported releases of GPFS. This support is introduced with client modification level 0.6 (PTF 6) of Version 3.1.

First, we describe differences between GPFS and JFS with respect to ADSM. Then, we discuss various resource requirements of ADSM for GPFS, which then lead to various configuration case studies. Finally, we summarize the results and our experience with different implementation configurations.

The goal of this chapter is to find answers to the following typical implementation questions:

- Which SP node should be used for the ADSM server? Does it matter if one runs the ADSM server and a VSD server on the same node?
- Which SP node should be used for the ADSM client? Is there any advantage of running the ADSM server and client on the same node?
- Is there any advantage to running multiple ADSM clients?
- How fast is a back up from GPFS? How fast can one restore to GPFS? How does it compare to JFS data?
- Can ADSM handle large file systems and large files in GPFS?

5.1 ADSM relevant differences between JFS and GPFS

Data streaming was the target for the design of Tiger Shark, the ancestor of GPFS. Since then, many modifications have been made to the file system structure to integrate this distributed file system into the parallel environment of the RS/6000 SP. As a result, there are some differences between JFS and

GPFS in terms of function, performance, and data volumes. This section describes the impact of these differences to ADSM.

5.1.1 Functional differences

GPFS and JFS are implemented as standard AIX Virtual File Systems. Although GPFS is a POSIX-compliant file system, some exceptions apply to GPFS. Table 18 summarizes functional differences between GPFS and JFS.

ACL Support:

AIX and GPFS have a different, even though related, implementation of access control lists (ACLs). The current ADSM client (Version 3.1.0.7) appears to back up GPFS ACLs as long as the user has restricted his or her usage of GPFS ACLs to the JFS subset of the standard, which is the vast majority of the function. It is planned to provide full ACLs support for GPFS in a future release of ADSM.

Table 18. ADSM relevant functional differences between JFS and GPFS

Function	JFS	GPFS 1.2
data location	single node	multiple nodes
locks	mandatory	advisory
memory mapped files	yes	no
atime, mtime, ctime	accurate	update delayed

One obvious difference between JFS and GPFS is the location of access and data. On JFS (no NFS), files can only be accessed from one machine; whereas, on GPFS, files can be accessed from all SP nodes where the GPFS file system is mounted. On JFS, files are located on local disks; whereas, on GPFS, files are spread across several disks on different VSD servers.

In GPFS, locks are advisory, which is the POSIX standard for locking files. Applications must explicitly check whether a file in GPFS is locked. The current release of the ADSM backup-archive client (V3.1.0.7), however, does not check for locks on files. As a result of this, ADSM will back up a file even when it is locked by another application. The same applies to the ADSM restore of a file that already exists in the GPFS file system and is locked by an application. ADSM will not check whether this file is locked.

The missing support for memory mapped files on GPFS has no impact to ADSM.

In GPFS, the change of a file's content and the change of its size are immediately propagated to all other SP nodes. However, there is a delay in the update of the atime, mtime, and ctime time stamps. When a file is accessed or changed on one node, it then can take GPFS up to five minutes to update the corresponding time stamps on different nodes. The delayed update of mtime has several impacts to ADSM which we discuss in more detail in the following section.

5.1.2 Impact of the delayed update of mtime

The delayed update of mtime may impact ADSM backup modes, its handling of files, which are modified during backup operation (serialization), and ADSM restore. This can easily be avoided by scheduling the ADSM operations appropriately. We recommend to allow sufficient time between user activity and scheduled ADSM backups and to discourage users from being active while ADSM operations are running. This would help to avoid or limit the following scenarios.

Backup operation

If a file has changed on one SP node and is backed up on a different node before mtime is updated on that node, then:

- Selective backup backs up the new content of the file, but it also backs up the old mtime.
- Incremental-by-date backup does not back up the file.
- Full incremental backup only backs up the file when either its size or its permissions have changed. If the backup occurs, ADSM backs up the new content of the file, however, it also backs up the old mtime. This is especially critical for the backup of databases since databases often change the content of their data files without changing their size or permissions.
- If the wrong mtime is backed up along with the file data to the ADSM server a succeeding full incremental backup will again back up this file, this time with the updated mtime, regardless as to whether the file contents have been changed.

Serialization

All ADSM backup modes use mtime and the size of files for serialization. When ADSM backs up a file, and at the same time the file is modified on a different node without changing its size, then ADSM will not try to send the file again when the backup of the file is finished before its mtime is updated. This means that, for such files, ADSM makes fuzzy backups even when

serialization is set to static. If a file is restored that contains a fuzzy backup, the file might not be usable depending on the file's application.

Restore operation

The delayed update of mtime can have three impacts to ADSM restore:

- If a file was backed up with the wrong mtime, ADSM will restore this mtime along with the data during a restore operation.
- Point-in-time restore of a file may fail when its backup copy is associated with an old mtime on the ADSM server.
- The restore operation of a file using the IFNEWER option may fail if the file is changed shortly before the restore operation occurs. For example, when a user restores a file and modifies it on a different node, then a second restore with the IFNEWER option will replace the changed file if the mtime on the local node is not yet updated.

5.1.3 Performance differences

GPFS is a shared disk file system that is optimized for the sequential access of large files. In comparison to JFS, the maximum I/O rate of GPFS is not limited by the maximum local I/O rate of a single file server. As a result, the maximum I/O rate of GPFS is much higher than the I/O rate of ADSM, which is also limited by the maximum local I/O rate of the ADSM server machine.

The GPFS metadata (inodes) is distributed over several SP nodes. Since GPFS must synchronize concurrent access from different nodes to the distributed metadata, the performance of accessing GPFS metadata is slower than the performance of JFS where the metadata is kept and accessed on a single machine. The slower performance of GPFS metadata access downgrades the performance of ADSM operations which access metadata of many files, such as incremental backup or restore of complete file systems.

As GPFS is originally designed for accessing large files, the performance for the access of a large number of small files is much worse than on JFS. For large files, GPFS can compensate the additional overhead of opening a file in a shared disk file system by striping large files over several VSD servers. Unfortunately, GPFS cannot benefit from striping several small files over different VSD servers.

5.1.4 Data volumes

The data volumes on GPFS file systems are often much higher than on JFS. GPFS currently supports file systems up to 5 TB with a maximum file size of 885 GB. ADSM can handle these data volumes; however, you must plan your

ADSM implementation carefully to be able to back up and restore those huge data volumes in a reasonable amount of time.

5.2 Resource requirements

A well tuned ADSM implementation for GPFS must take care of the resource requirements of all components of ADSM and GPFS. To plan an implementation of ADSM for GPFS, resource requirements in terms of CPU capacity, memory capacity, device I/O rate, and network bandwidth must be taken into account. Table 19 summarizes the resource requirements of the various components of GPFS and ADSM.

Table 19. Resource requirements for GPFS and ADSM components

	CPU Usage	Memory Usage	Disk I/O	Network I/O
VSD server	low to medium	high	high	medium to high
VSD client on ADSM client node	medium	low	none	medium to high
Configuration Manager	very low	very low	none	very low
Stripe Group Manager/ Token Manager server	medium to high	high	none	medium to high
Metadata Manager and Token Manager on ADSM client node	low to medium	low	none	low to medium
ADSM server	medium to high	high	high	medium to high
ADSM client	low	high	none	low to medium

In the following, we give reasons for the resource requirements shown in Table 19 by the means of the example configuration shown in Figure 33 on page 170. It shows a GPFS cluster running on four SP nodes. Two VSD servers, which are connected via twin-tailed cabling, provide the storage for GPFS. Two ADSM clients back up the GPFS file systems to a single ADSM server.

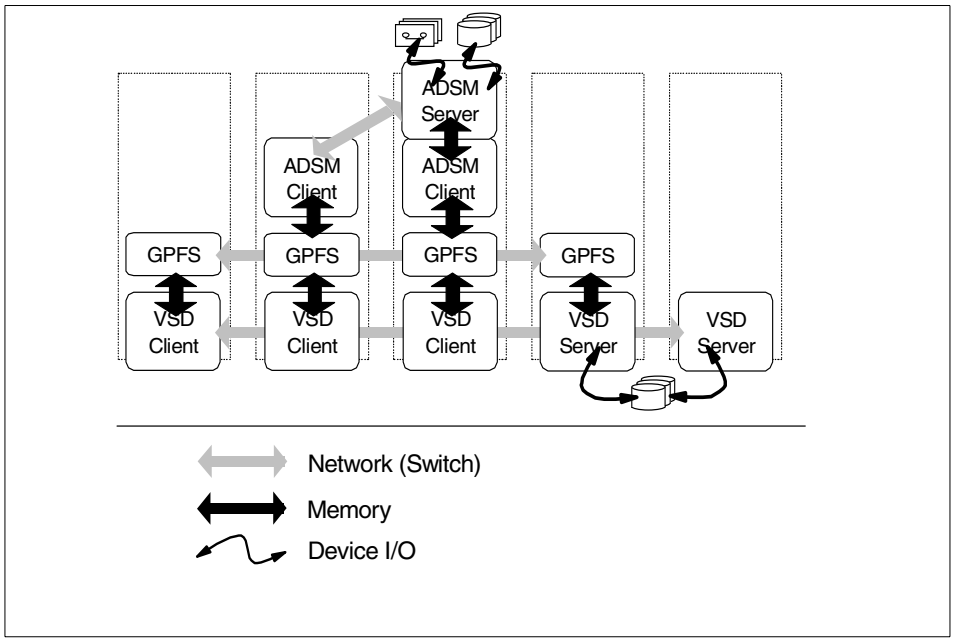


Figure 33. Example implementation for ADSM on GPFS

5.2.1 Resource requirements for VSD

Independent of ADSM, VSD server for large GPFS file systems require usual end user activity to be low to medium CPU power and have a high usage of all other resources. Due to the immense resource requirements, it is generally recommended to have dedicated VSD server nodes for large GPFS file systems. For example, the node on the right in Figure 33 does not even run the GPFS daemon.

VSD clients require medium CPU power and medium to high network bandwidth depending on the GPFS file system activity generated on this node. Of course, ADSM backup and restore operations generate high file system activity on SP nodes where the ADSM client is running. Notice that VSD clients do not require I/O bandwidth. They only consume I/O bandwidth on their associated VSD server nodes.

5.2.2 Resource requirements for GPFS

In our example, the GPFS daemon is running on four SP nodes. Each node's GPFS daemon can assume different personalities. The resource

requirements of a GPFS daemon depend on its taken personalties. Only the GPFS Configuration Manager personality requires nearly no resources.

Each GPFS file system has a Stripe Group Manager and Token Manager Server, where both services must run on the same SP node. For large GPFS file systems, both personalities require medium to high CPU capacity, high memory capacity, and medium to high network I/O bandwidth. Therefore, we recommend running the Stripe Group Manager and ADSM backup sessions of large GPFS file systems on dedicated nodes.

Each access to a file requires interaction with the node's GPFS Token Manager (which interacts with the corresponding GPFS Token Manager Server) and with the file's Metadata Manager. In almost all cases, the node that has opened a certain file for the longest period of continuous time, is the file's Metadata Manager. Since the ADSM client accesses a lot of files for back up and restore, it increases the resource requirements of its local GPFS daemon by low to medium CPU capacity and by low to medium network bandwidth.

5.2.3 Resource requirements for ADSM

The ADSM client itself requires high memory capacity and low to medium network bandwidth for back up to, or restore from, the ADSM server. In addition to this, ADSM client activity also increases the resource requirements of its node's VSD client and the GPFS daemon as described above. Therefore, the total resource requirement of the SP node running the ADSM client is medium to high for CPU capacity and network bandwidth and high for memory capacity.

The ADSM server's memory and the device I/O bandwidth required is high. The required network I/O bandwidth is medium to high depending on the number of concurrent ADSM client sessions. The ADSM server requires medium CPU power for back up and restore, but for ADSM internal processes, for example, the expiration of the inventory, the CPU usage is high.

5.3 Case studies

In this section, we first describe the hardware and the software of our test environment, define our test methodology, and document the settings for ADSM, TCPIP, and GPFS used throughout the testing.

We then describe our experience with four different example configurations. All configurations share the same setup for the ADSM server and the VSD

servers. The configurations differ by the location and number of the ADSM clients only.

We then depict the impact of tuning maxFileToCache on the data throughput and compare JFS and GPFS performance data.

Finally, we document functional tests and their results to compare ADSM operations. We compare the performance results of incremental backup against selective backup, and also restore against restore, using the replace option.

5.3.1 Test system configuration

The different configurations were tested on a RS/6000 SP with eight Silver Wide nodes running AIX 4.3.1, each with four 332 MHz 640e CPUs and 3 GB memory. The tests ran on ADSM server 3.1.2.20, ADSM client 3.1.0.7, GPFS 1.2, and PSSP 3.1.1.3.

The ADSM server ran on a dedicated SP node. The ADSM server's recovery log and database volumes were stored on internal disks. The ADSM server's storage pool volumes were located on four 7133-D40 disks of size 9.1 GB connected to an IBM Advanced Serial SSA RAID adapter (feature code 6225). Figure 34 shows the cabling of the disks. On each disk, a JFS file system with one storage pool volume was created.

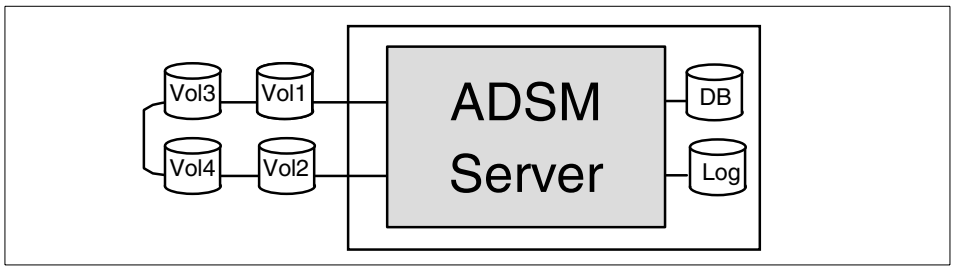


Figure 34. Cabling and volumes of ADSM Server

The storage for the GPFS test file system was provided by two pairs of twin-cabled VSD server nodes, each equipped with one SSA 6225 adapter. Each pair of VSD servers was cabled as shown in Figure 35 on page 173. In total we had twenty-four SSA disks, each of size 9.1 GB. The location of the GPFS daemons and the ADSM clients varied for the different configurations.

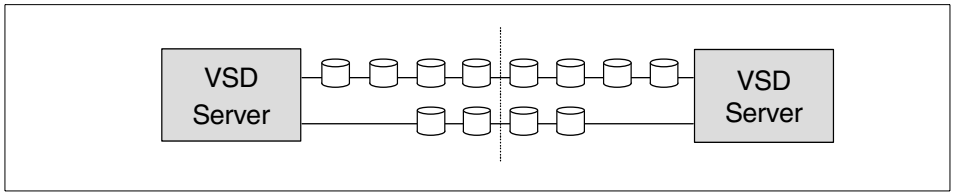


Figure 35. Cabling of VSD servers

The performance was measured for backup and restore with different file sizes and up to four concurrent ADSM client sessions. Each ADSM client session had its own directory with test files as given in Table 20.

Table 20. Sizing of test data

File Size	Number of Directories	Number of Files per Directory
256 M	1	4
10 M	3	25
1 M	10	125
100 K	20	150
10 K	10	150
1 K	10	135

5.3.2 Test methodology

The results for these tests were obtained by running one or more ADSM client sessions on one or more of the SP nodes.

Each ADSM client session operated on a separate subdirectory tree within the same JFS or GPFS file system. The structure of the directory tree containing the test data and the number of files are described in Table 20.

The following operations were performed for each test:

1. Full incremental backup
2. Restore to an alternative directory within the same file system
3. Restore to the same alternate directory within the same file system. The REPLACE=YES option was specified.

The contents of the alternative restore directory were deleted before the first restore operation but not before the second restore operation.

All client file spaces were deleted on the ADSM server between successive runs to ensure that there was no effect on performance over time due to the contents of the ADSM database or log.

The ADSM server was restarted, and the JFS file systems containing the disk storage pool volumes were unmounted and re-mounted before performing each restore operation. This ensured that all data being restored was actually read from disk and not just cached from memory.

5.3.3 ADSM and TCP/IP configuration

This section documents the TCP/IP and ADSM client and server options used during the running of the tests.

ADSM server options

The ADSM server options used during all tests are shown in Table 21.

Table 21. ADSM server options

Parameter	Value
TCPWindowsize	256
TCPBufsize	32
TCPNodelay	yes
TXNGroupmax	256

ADSM client options

The ADSM client options used for all ADSM client nodes on all tests are shown in Table 22.

Table 22. ADSM client options

Parameter	Value
LARGECOMmbuffers	yes
TCPWindowsize	256
TCPBufsize	32
TCPNodelay	yes
TXNBytelimit	25600

TCP/IP options

The TCP/IP options used on all SP nodes are shown in Table 23.

Table 23. TCP/IP parameters for all SP Nodes

Parameter	Value
thewall	65536
sb_max	1310720
tcp_sendspace	327680
tcp_recvspace	327680
tcp_mssdflt	32768
rfc1323	1

GPFS file system options

The GPFS file system option used during the testing are shown in Table 24. Some tests were also done to measure the effect of increasing maxFilesToCache.

Table 24. GPFS file system parameters

Parameter	Value
mallocsize	20 MB
maxFilesToCache	200
Block Size	256 KB
I-node Size	512
Indirect Block Size	16 KB
Stripe Method	Round Robin

5.3.4 Configuration 1: ADSM client on single VSD client node

Configuration 1 is called *Reference Configuration* for two reasons: First, this configuration is similar to the configuration for local file systems, such as JFS, where the ADSM clients cannot be distributed on several nodes. Second, in the following subsections, performance of the other three configuration is shown in comparison to the performance of Configuration 1.

Configuration description

Figure 36 on page 176 shows Configuration 1, the Reference Configuration. The ADSM client runs on a VSD client, which is separate from the ADSM

server. When running multiple concurrent ADSM client sessions, all sessions are started on the same SP node.

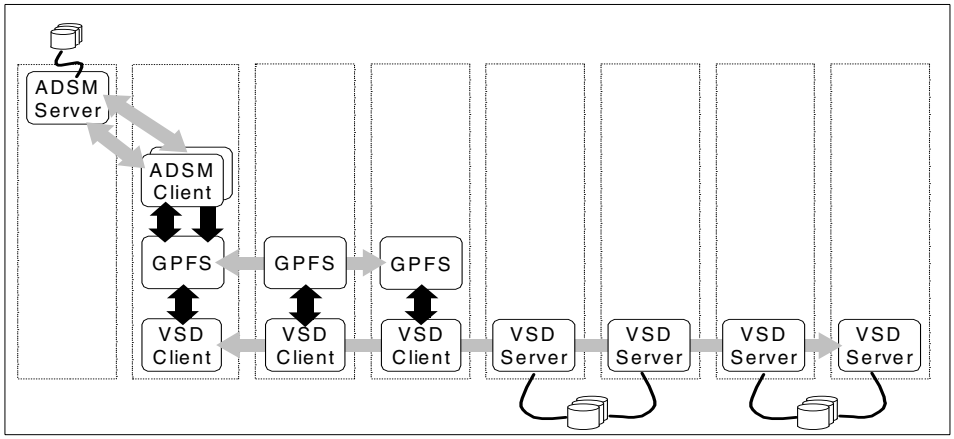


Figure 36. Multiple ADSM Client Sessions on single VSD Client Node

Test results

The results for this configuration are used in the following sections to compare and contrast the performance differences between the various configurations. Table 25 lists the throughput numbers for backup and restore which are also illustrated in Figure 37 on page 178 and Figure 39 on page 179. Figure 38 on page 178 and Figure 40 on page 179 show the ADSM server CPU utilization during the corresponding ADSM operations.

Table 25. Backup and restore throughput for reference configuration

File Size	# of Client Sessions	Data Rate (KB/s)	
		Backup	Restore
256 MB	1	8,852.73	13,291.99
	2	16,036.87	25,922.52
	4	25,761.17	36,442.80
10 MB	1	8,385.9	11,566.57
	2	15,664.07	22,087.69
	4	23,270.83	30,256.15

File Size	# of Client Sessions	Data Rate (KB/s)	
		Backup	Restore
1 MB	1	4,217.10	4,555.00
	2	7,349.75	8,605.39
	4	10,938.46	14,713.25
100 KB	1	901.12	786.80
	2	1,241.59	1,381.30
	4	1,714.36	2,159.85
10 KB	1	99.27	83.42
	2	115.65	158.56
	4	169.09	231.61
1 KB	1	10.25	8.49
	2	11.32	16.23
	4	16.90	23.31

The following are reference configurations.

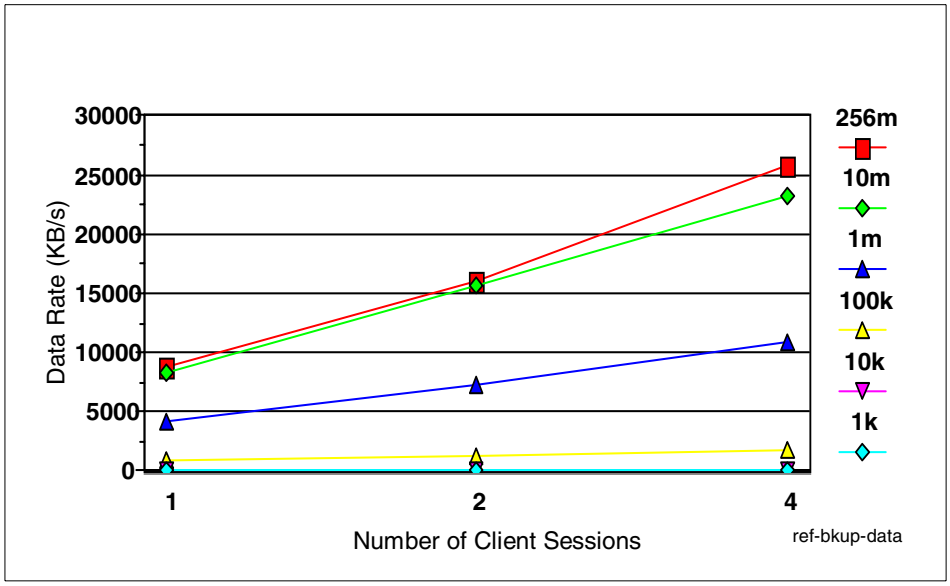


Figure 37. Backup data throughput

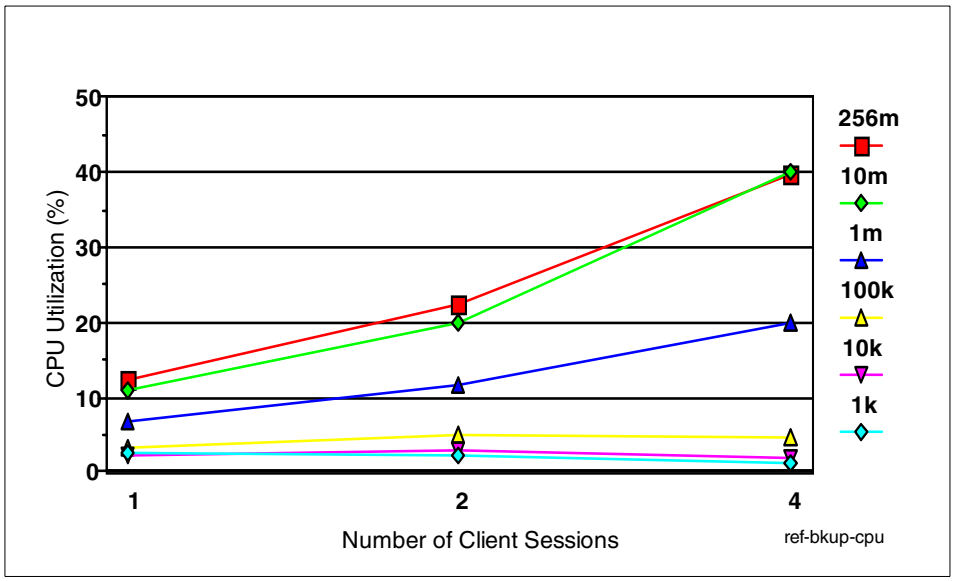


Figure 38. Reference Configuration: ADSM Server CPU Utilization - Backup

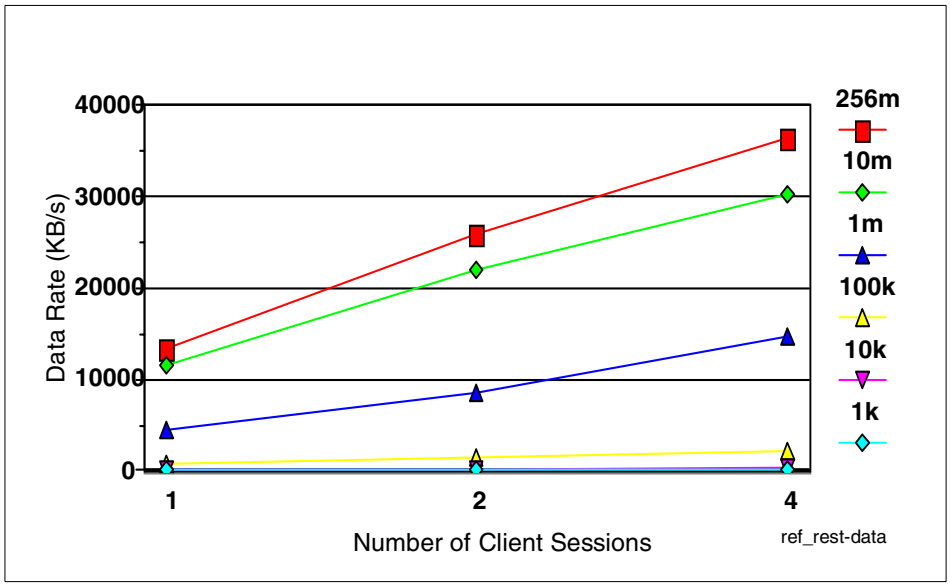


Figure 39. Restore data throughput

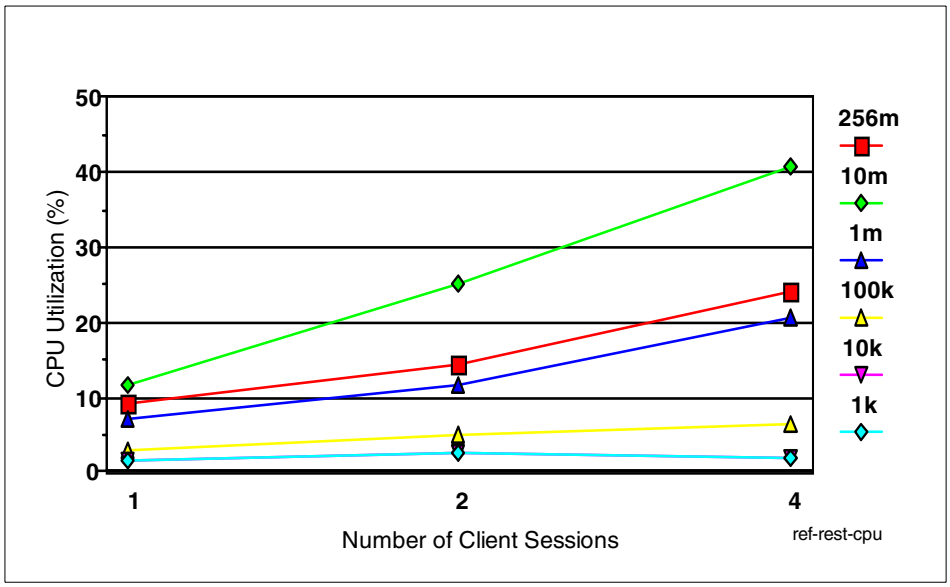


Figure 40. ADSM server CPU utilization - Restore

5.3.5 Configuration 2: ADSM client and server on same SP node

This test was performed to determine whether or not shared memory communication between the ADSM server and client would improve performance.

Configuration description

In Configuration 2, as illustrated in Figure 41, up to four ADSM client sessions run on the ADSM server node. Configuration 2 checks whether ADSM can benefit from using shared-memory communication between ADSM server and ADSM client.

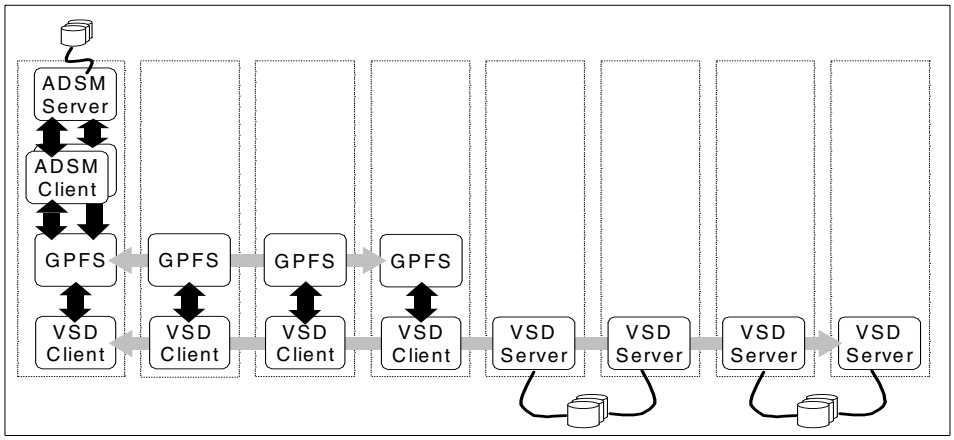


Figure 41. Configuration 2: Multiple ADSM client sessions on ADSM server node

Test results

The data transfer rates for full incremental backup and restore are shown in Table 26. The data transfer rates are expressed relative to the measurements made on the Reference Configuration. The measurements for 256 MB files and 10 KB files are also shown in Figure 42 on page 182 and Figure 43 on page 182. Figure 44 on page 183 illustrates the comparison of ADSM server CPU utilization using shared memory and TCP/IP communication.

The results show that there is little benefit for large files in running the ADSM server and client on the same SP node. For smaller files, there is some benefit in running the ADSM server and client on the same node but only when running one or two client sessions. When running three or more sessions, the CPU utilization becomes a limiting factor.

Table 26. Relative data throughput using TCP/IP and shared memory

File Size	# of Client Sessions	Backup		Restore	
		TCP/IP	ShMem	TCP/IP	ShMem
256 MB	1	1.0267	0.984	0.992	0.992
	2	0.940	0.936	0.902	0.930
	4	0.917	0.913	1.076	0.752
10 MB	1	0.977	0.975	0.999	0.990
	2	0.925	0.942	0.959	0.963
	4	0.913	0.900	0.857	0.849
1 MB	1	1.004	1.004	0.995	1.000
	2	1.054	1.057	0.986	0.982
	4	0.878	0.907	0.965	0.948
100 KB	1	1.125	1.021	1.029	0.983
	2	1.069	0.968	1.054	1.006
	4	1.023	0.887	0.994	0.957
10 KB	1	1.180	1.091	1.049	0.999
	2	1.156	1.038	1.052	0.992
	4	0.986	0.875	1.035	0.974
1 KB	1	1.150	1.033	1.066	0.998
	2	1.180	1.045	1.051	0.994
	4	1.054	0.934	1.005	0.969

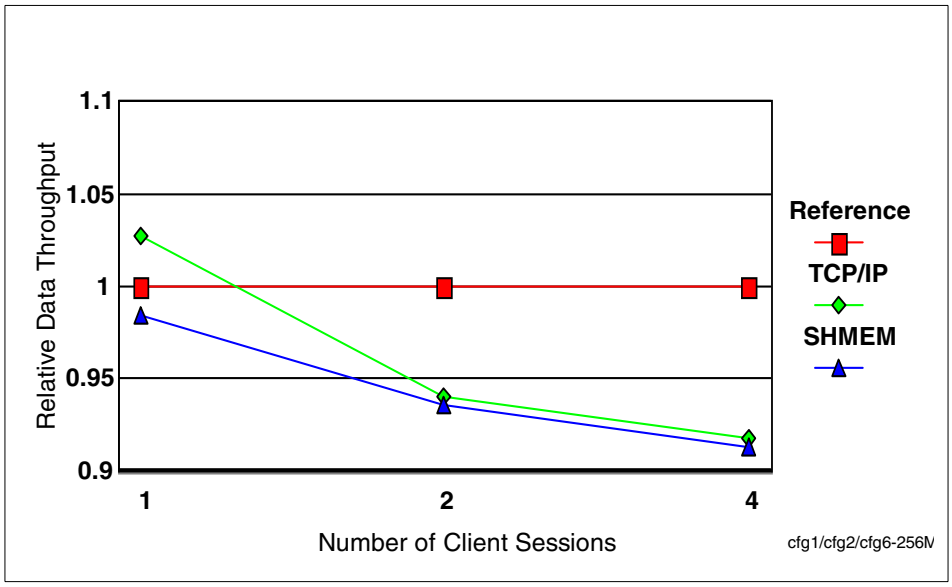


Figure 42. Rel. backup data throughput: 256 MB files (IP and shared memory)

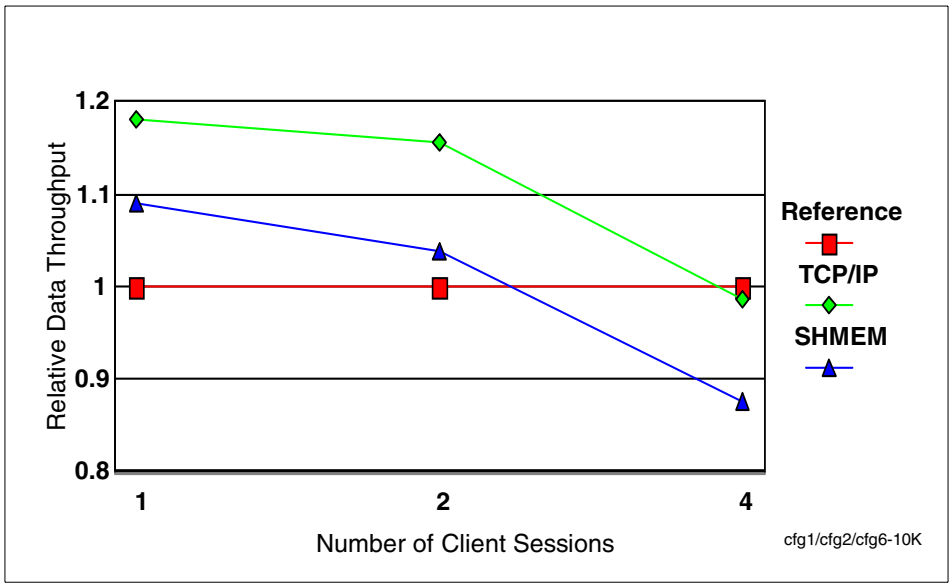


Figure 43. Rel. backup data throughput: 10 KB files (IP and shared memory)

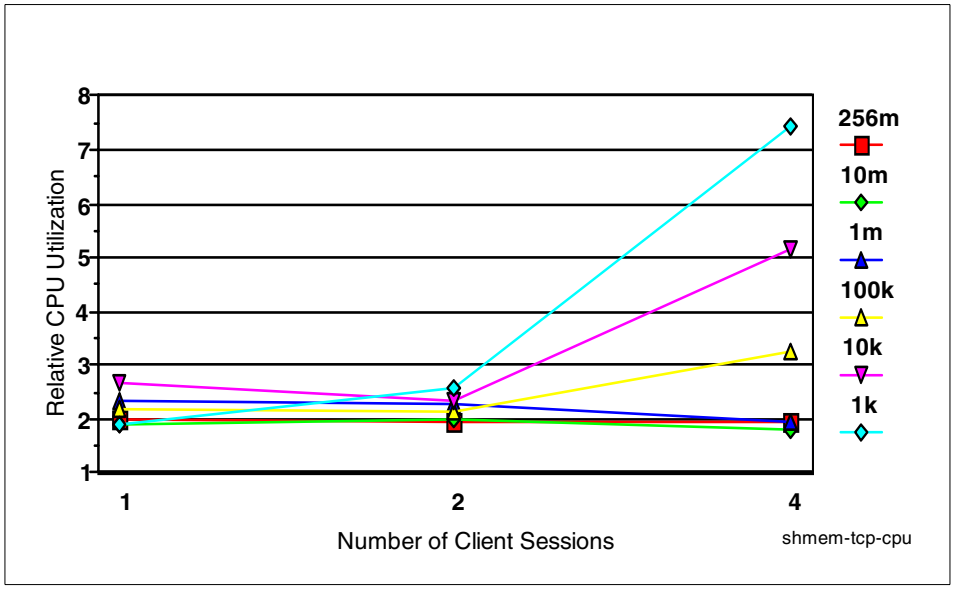


Figure 44. ADSM server CPU utilization — Backup (shared memory to IP)

5.3.6 Configuration 3: Using multiple ADSM client nodes

These tests were performed to determine what benefit could be obtained by using multiple SP nodes for the ADSM client sessions. The first test used four separate SP nodes to run one, two, and four client sessions. The second test used two SP nodes for the same number of client sessions.

Configuration description

Configuration 3, as illustrated in Figure 45, tests whether ADSM can benefit from the infrastructure of the parallel RS/6000 SP environment. Like in Configuration 1, the ADSM clients are running on VSD client nodes that are not the ADSM server, but now the ADSM clients are running on different SP nodes. For running four ADSM client sessions, we have tested two variations. In variation 3a, all ADSM clients are running on separate SP nodes. In variation 3b, the ADSM clients are running on two SP nodes with up to two ADSM client sessions on each node.

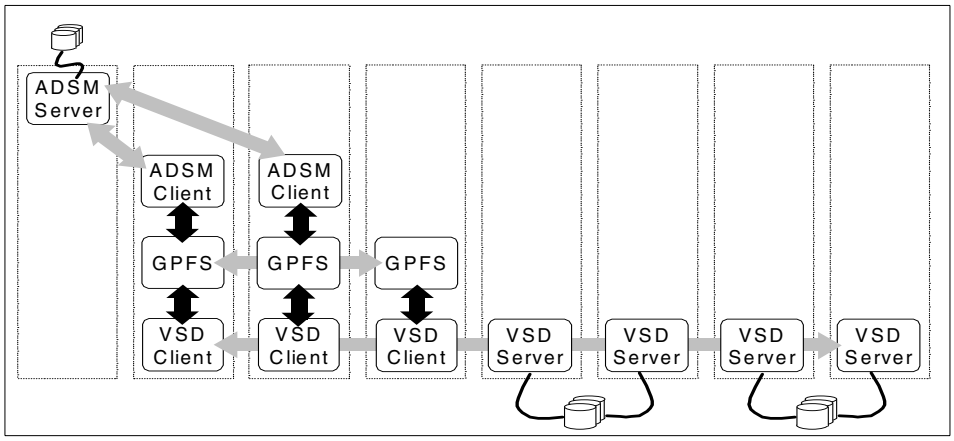


Figure 45. Configuration 3: Multiple ADSM/VSD client sessions and nodes

Test results

The first test (3a) was running ADSM clients on three VSD client nodes and one VSD server node (our system setup was limited to three VSD client nodes). The data throughput measurements for full incremental backup relative to the Reference Configuration are shown in Figure 46 on page 185, and for restore in Figure 47 on page 185.

The second test (3b) was running ADSM clients on two VSD client nodes. The data throughput measurements for full incremental backup relative to the Reference Configuration are shown in Figure 48 on page 186, and for restore in Figure 49 on page 186.

The results show that there is little improvement for large files in running the ADSM client on more than one SP node. This indicates that the ADSM client software and the SP node itself are not the limiting factors when transferring large volumes of data. The improvement for small files, however, is significant.

For restore operations, there is a similar benefit in using multiple ADSM client nodes but only for very small files.

When the number of ADSM client nodes is limited to two, we see a similar improvement in performance for backing up small files. This effect is reduced when we run more than one client session on the same SP node.

The performance for restore operations on two SP nodes is also reduced when more than one client is run on the node.

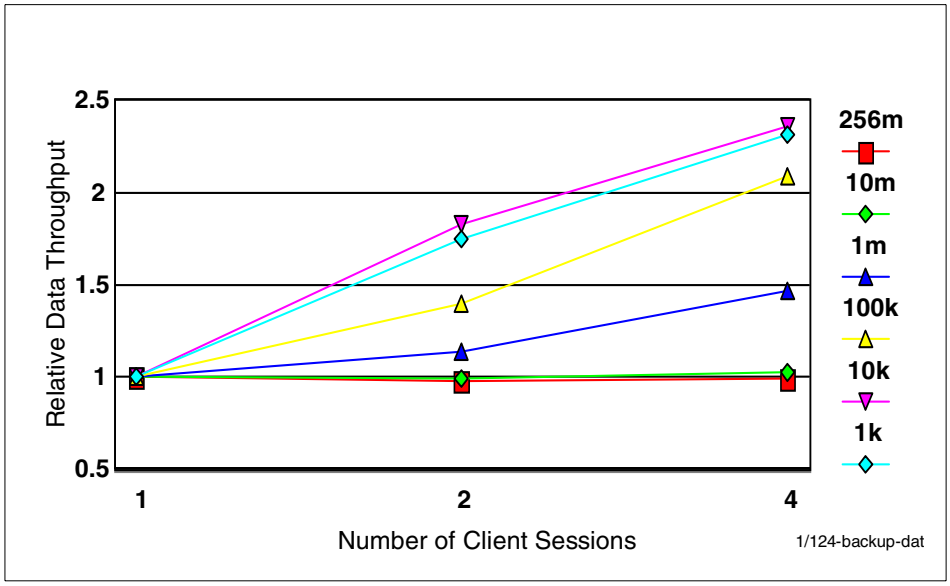


Figure 46. Rel. backup data throughput: ADSM client running on four SP Nodes

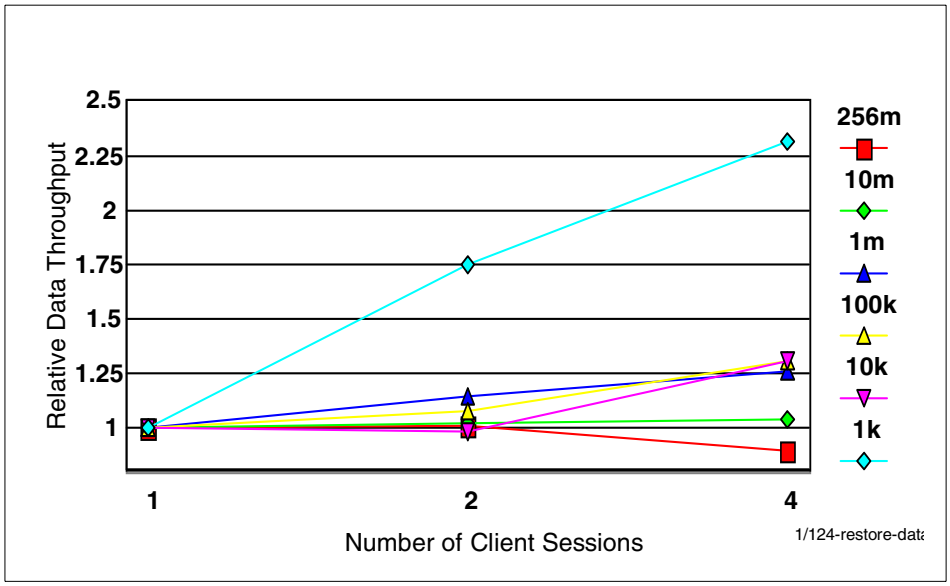


Figure 47. Rel. restore data throughput: ADSM client running on four SP Nodes

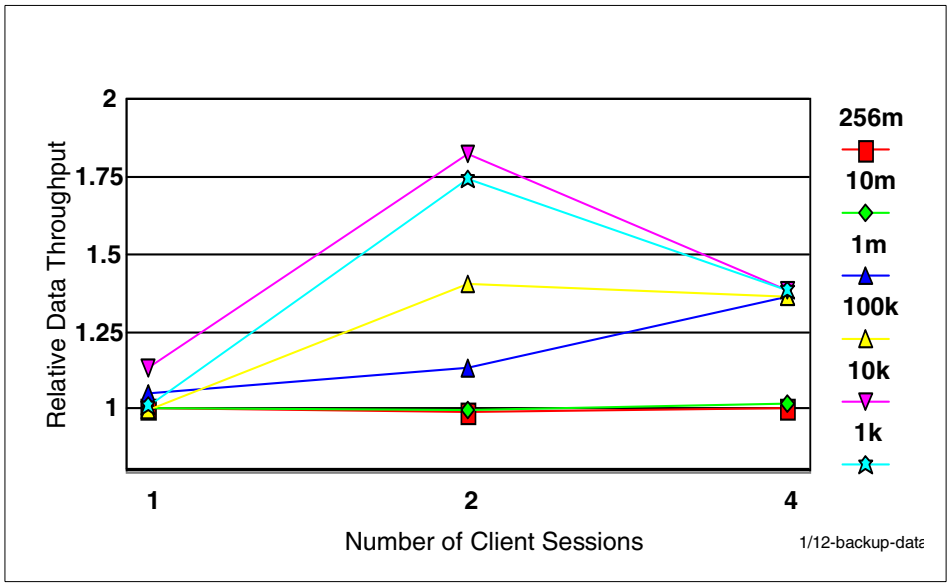


Figure 48. Rel. backup data throughput: ADSM client running on two SP Nodes

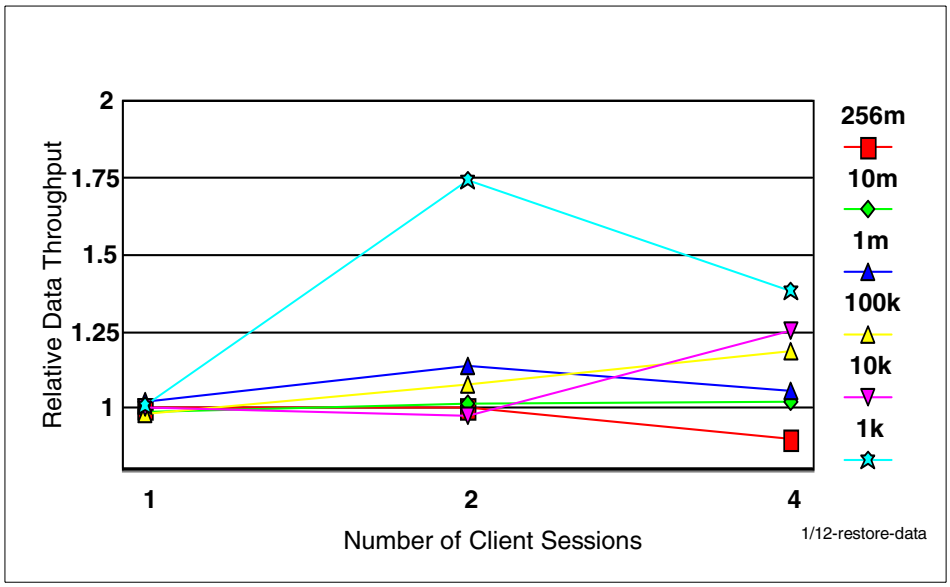


Figure 49. Rel. restore data throughput: ADSM client running on two SP Nodes

5.3.7 Configuration 4: ADSM clients on VSD server nodes

The purpose of this test was to determine whether there would be any benefit in running the ADSM client local to some of the data and to see whether there would be any contention for resources between the ADSM client and the VSD server.

Configuration description

Configuration 4, as illustrated in Figure 50, tests whether there is a benefit or a disadvantage of running the ADSM clients on VSD server nodes. We have retested Configuration 1 and Configuration 3b, but this time with the ADSM client sessions running on VSD server nodes.

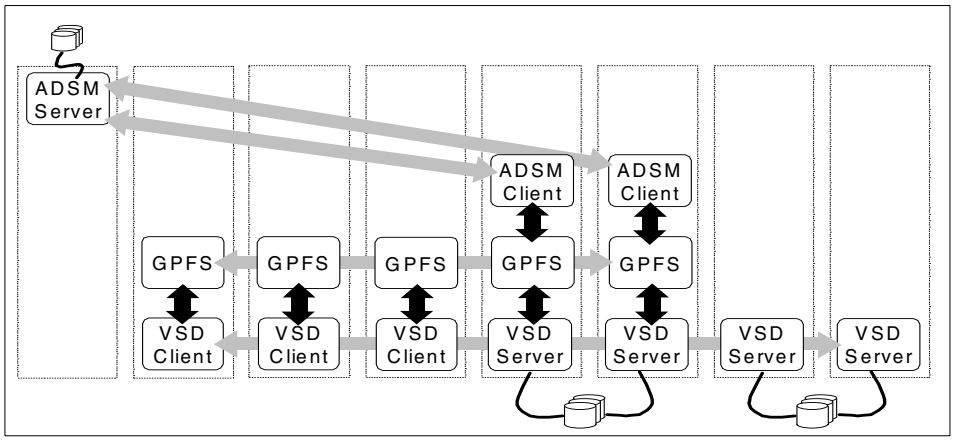


Figure 50. Conf. 4: Multiple ADSM client sessions on multiple VSD server nodes

Test results

The full incremental backup performance relative to the Reference Configuration for a single ADSM client on a VSD server is shown in Figure 51 on page 188. Figure 52 on page 188 shows the same performance information for two ADSM clients on two VSD servers.

The results show that while there is no significant benefit in running the ADSM client on the VSD server node, there is also little interference.

When two ADSM clients are run on separate VSD servers, the result is very similar.

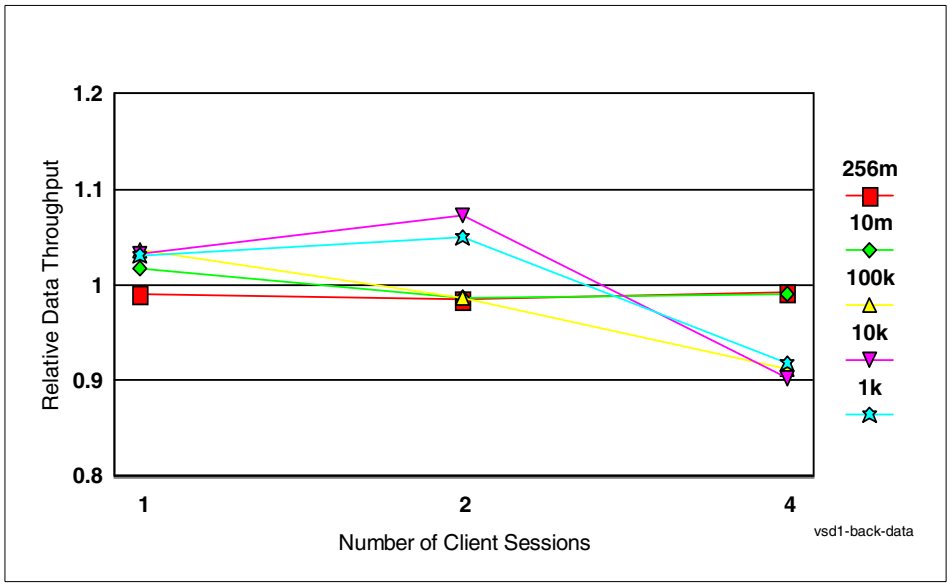


Figure 51. Rel. backup data throughput: ADSM client on one VSD server

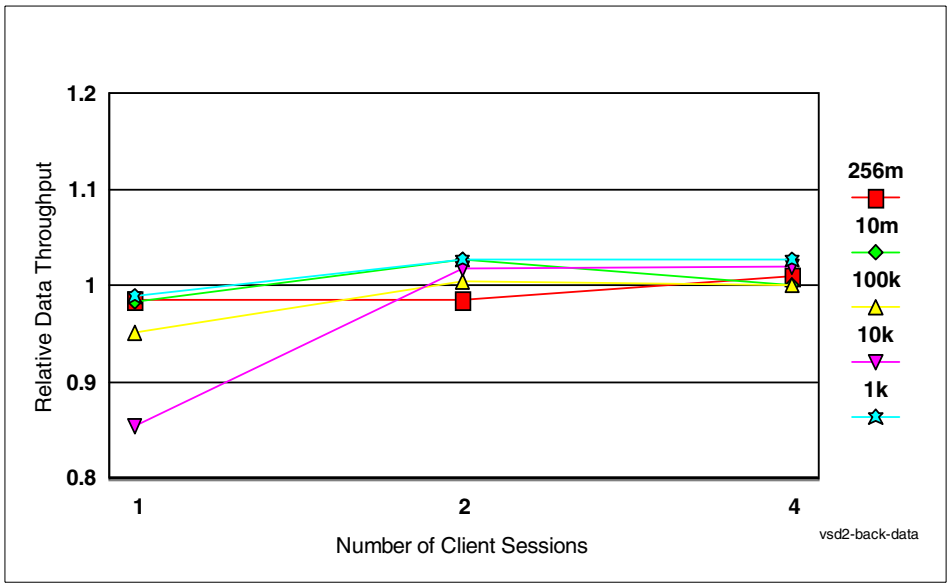


Figure 52. Rel. backup data throughput: ADSM client on two VSD servers

5.3.8 Impact of tuning maxFilesToCache

ADSM performance is closely linked to the speed at which file metadata and inodes can be accessed. The value of maxFilesToCache was increased to see whether any improvement in performance would be obtained.

In the first test, no data was sent from the ADSM client to the ADSM server. It involved performing a full incremental backup on a subdirectory containing 20,000 files, which had already been backed up and not changed since. The ADSM client sessions were started from one single SP node.

The time taken to traverse the subdirectory tree for various settings of maxFilesToCache and mallocsize is shown in Table 27. The time taken to perform the same operation on a JFS file system is also shown for comparison.

The effect of increasing maxFilesToCache during a full incremental backup and a restore operation is shown in Figure 53 on page 190 and Figure 54 on page 190.The results shown are relative to the Reference Configuration which used the default maxFilesToCache value of 200. The test was performed with maxFilesToCache set to 10,000.

Note that only half of the mallocpool can be used for cached inodes, and each inode caching takes up about 5.5 KB. Thus, with 40 MB mallocsize, maxFilesToCache is limited internally, and its effective value would be 3800 (the number of files cached is limited to one half of the mallocsize divided by 5.5 KB). The same adjustment applies to a mallocsize of 60 MB. Even with maxFilesToCache set to 21,000, the effective value would be 5600.

Table 27. Effect of maxFilesToCache on full incremental backups

File System Type	maxFilesToCache	mallocsize	Time to Perform Full Incremental
JFS	n/a	n/a	00:00:14
GPFS	200	20 MB	00:05:08
	10,000	40 MB	00:03:26
	21,000	60 MB	00:02:51

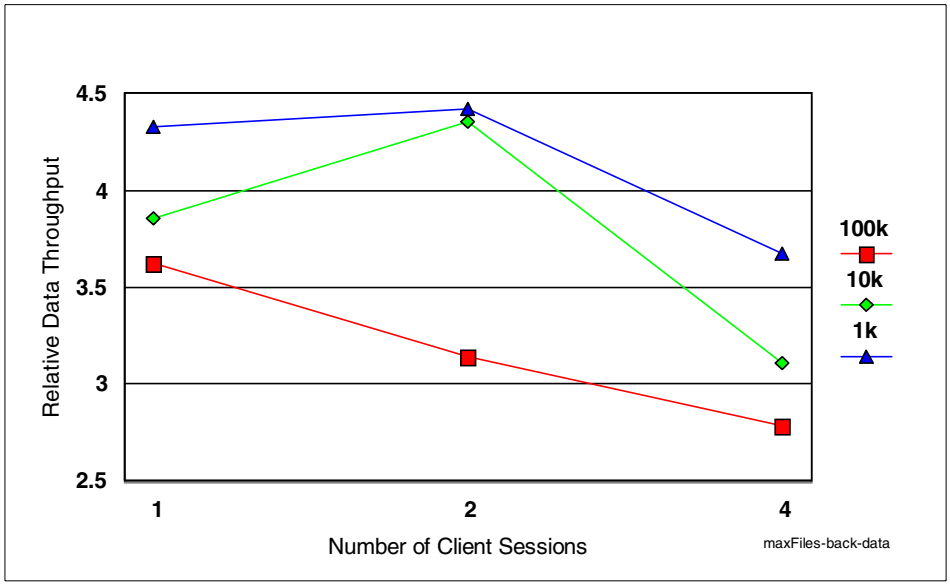


Figure 53. Backup data throughput: Ratio of maxFilesToCache 10,000 to 200

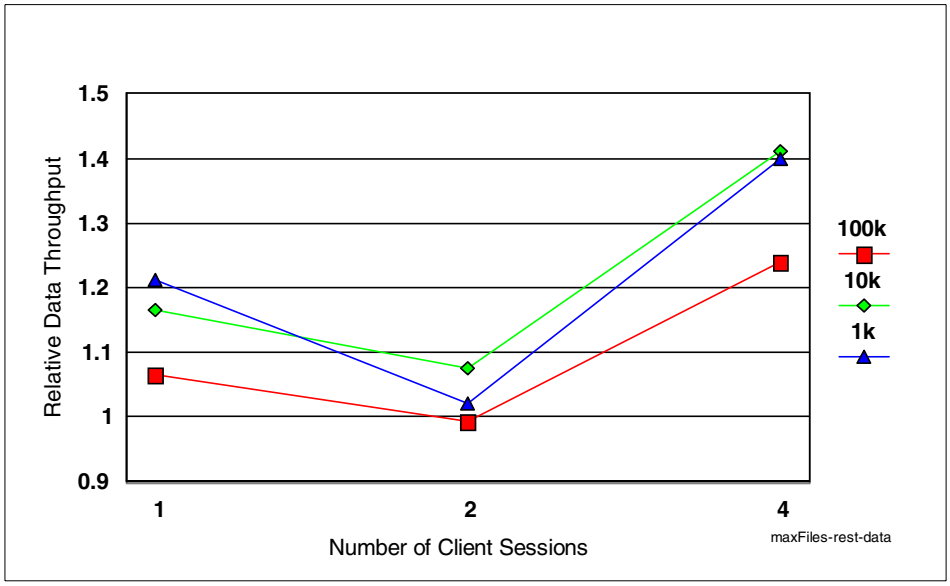


Figure 54. Restore data throughput: Ratio of maxFilesToCache 10,000 to 200

5.3.9 Comparison of ADSM performance between JFS and GPFS

In this section, we compare some of the results we obtained during the previous case studies on GPFS with results from similar tests where the data resides on JFS.

The results illustrated in Figure 55 show that, for small files, backup operations are significantly faster from JFS than GPFS.

Increasing the GPFS parameter maxFilesToCache reduces the differential between the two file systems. The default value for maxFileToCache is 200. The improvement obtained in increasing this value to 10,000 is shown in Figure 56 on page 192.

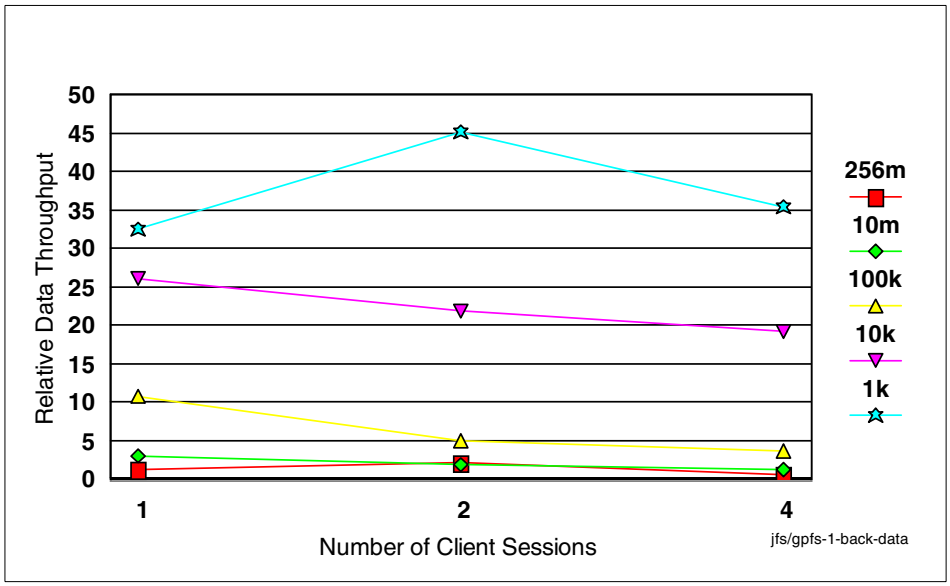


Figure 55. Backup data throughput: Ratio of JFS to GPFS

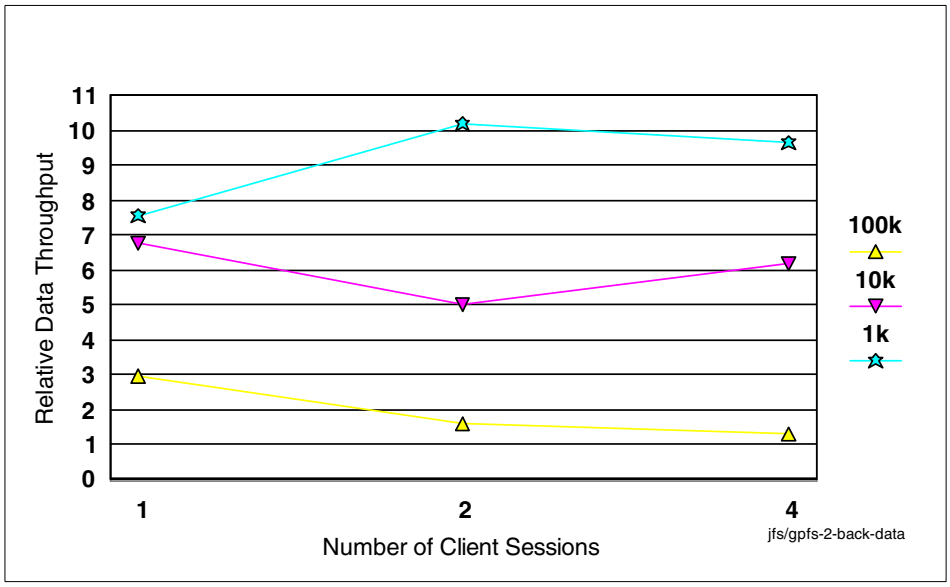


Figure 56. Backup data throughput: JFS/GPFS — Increased maxFilesToCache

5.3.10 Full Incremental versus selective backup

We know that accessing metadata in GPFS is slower than in JFS; so, the purpose of this test was to determine what effect this would have on full incremental backup performance compared to selective backup. Both backup operations were performed on the same data.

The results in Figure 57 show that selective backup is faster than full incremental backup and that the difference increases as the files get smaller.

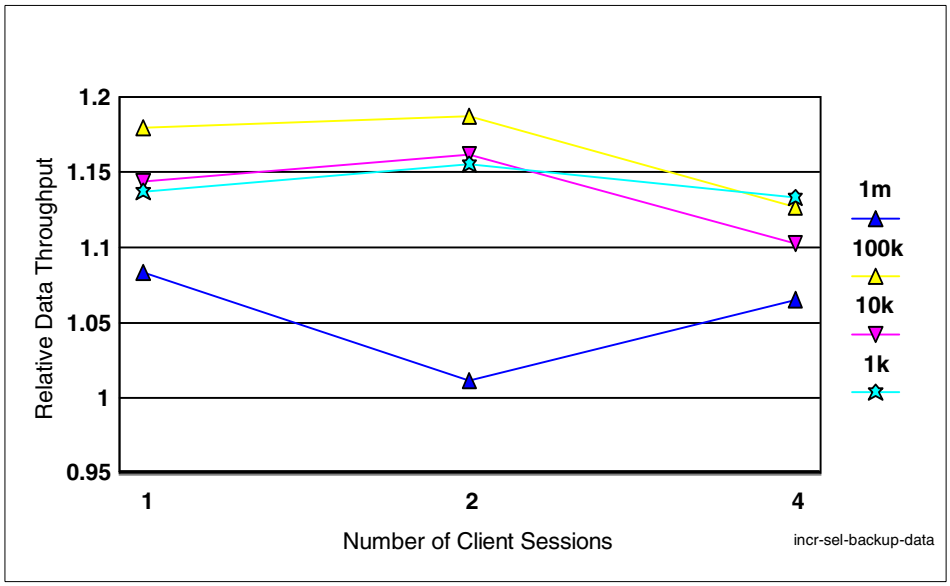


Figure 57. Backup data throughput: Ratio of selective to full Incremental

5.3.11 Restore versus replace

The purpose of this test was to determine whether there was any difference in behavior of GPFS when the file being restored already existed in the file system.

As was described in 5.3.2, “Test methodology” on page 173, each test involved restoring the files to an empty directory and then repeating the restore operation with the REPLACE=YES. Figure 58 on page 194 shows the comparison between restore and replace for the Reference Configuration. The graph shows the data throughput rate for a replace operation relative to the same operation performed with no replace.

For large files, there is little difference between the behavior of the restore and replace operations. But, as the size of the file decreases, the replace operation becomes increasingly faster. This effect is reduced as the number of ADSM client sessions running on the same SP node is increased.

The same comparison has been done when the multiple ADSM client sessions are running on separate SP nodes. The results for this test are shown in Figure 59 on page 195.

As can be seen, the results are quite different for the case where two and four client sessions are running. The improvement in performance seen with one client session is maintained across all sessions.

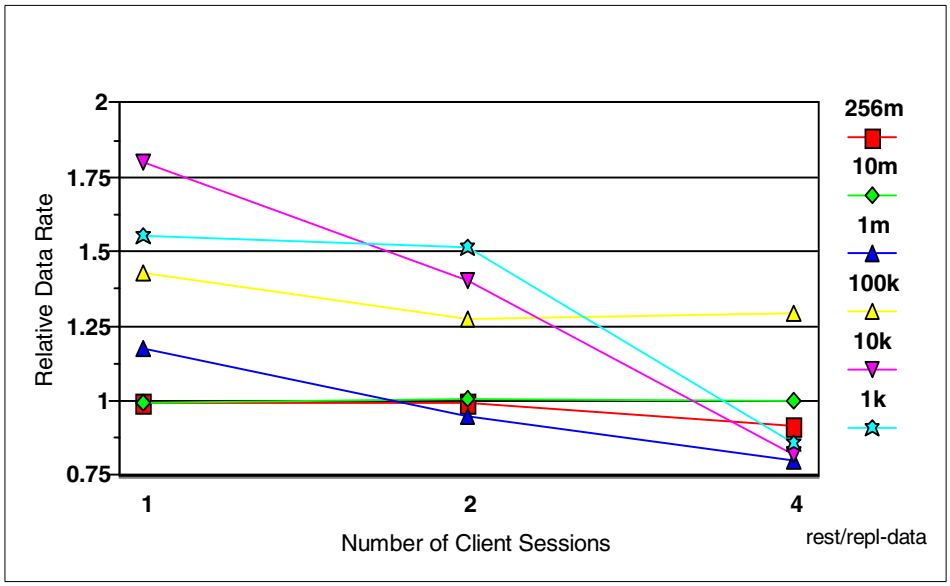


Figure 58. Restore data throughput: Ratio of replace to restore: Single SP node

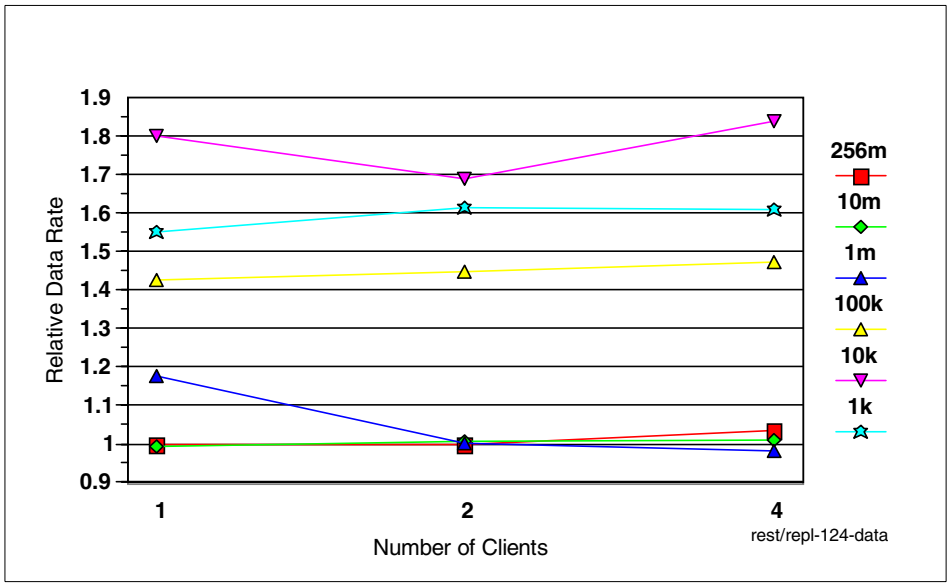


Figure 59. Restore data throughput: Ratio Replace/Restore: Multiple SP nodes

5.4 Recommendations

The following summarizes the results and our experience with different implementation configurations. The results obtained certainly are limited to our system environment and testing during the residency. Our answers and recommendations cannot satisfy the questions for all possible configurations. However, we think that at least they may help you, as discussion, during the implementation planning steps.

5.4.1 Which SP Node to use as an ADSM server

When looking for a location for the ADSM server, you first have to decide whether to put the ADSM server inside or outside the RS/6000 SP. Of course, there is no choice if you have to back up a new GPFS file system to an already established ADSM server. But, the bandwidth of the network between the ADSM server and the ADSM client may become a critical resource bottleneck.

Therefore, when setting up a new ADSM server for GPFS, you should locate the ADSM server on a RS/6000 SP node. Because of the high performance requirements of the ADSM server, you should consider having a dedicated

SP node for the ADSM server, which does not run any other resource intensive tasks, for example, ADSM client sessions or VSD server processes.

There is no real benefit from locating the ADSM server on a VSD server node because the data of a GPFS file system is spread over all VSD server nodes. We did not specifically test this constellation because we felt that there would be too many conflicts in resources to make this a worthwhile configuration. There are also likely to be problems in connecting large volumes of disk storage for GPFS and tape drives for ADSM on the same SP node.

5.4.2 Which SP node to use as an ADSM client

When the ADSM server is located inside the RS/6000 SP, then you have three options for the location of the ADSM client:

- An SP node only running GPFS but not the VSD server
- A VSD server node
- The ADSM server node.

Due to the resource requirements of all components, you may consider to run the ADSM client on an SP node that is neither the ADSM server node nor a VSD server node.

The results illustrated in Figure 42 on page 182 and Figure 43 on page 182 show that there is also no real benefit from shared-memory communication when ADSM client and ADSM server run on the same SP node. Inside the RS/6000 SP, the network bandwidth between ADSM server and ADSM client is no performance bottleneck because of the good network performance provided by the SP Switch.

However, collocating ADSM server and client may make sense because you then have dedicated one single SP node to ADSM instead of two or more. This gives you additional CPU and hardware on the other nodes to run your applications.

The results illustrated in Figure 51 on page 188 and Figure 52 on page 188 show that there is no significant advantage in running the ADSM client on a VSD server node, especially when using multiple client sessions, because the data of a GPFS file system is spread over all VSD server nodes. From the point of view of minimizing resource conflicts, it would be best to run the ADSM client on an SP node that is neither a VSD server nor a Stripe Group Manager.

5.4.3 Is there any advantage on running multiple client sessions?

The GPFS file system is capable of supplying data to a client application at very high data rates. In order to capitalize on this performance, it is necessary to run multiple ADSM client sessions from separate GPFS file systems or the same GPFS file system. This allows the ADSM server to use multiple storage pool devices to receive the data.

Using multiple ADSM client sessions within the same file system currently requires each session to target a specific subdirectory tree within the file system. Care should be taken to balance the volume of data between each client session.

If your backup data is sent directly to tape media, then care must be taken during the restore process to avoid client sessions requesting the same tape. This can be achieved by splitting the restore sessions across the same subdirectory arrangement used during the backup operation. If your directory structure is likely to change on a frequent basis, then this may not be feasible.

5.4.4 How many SP Nodes be used as an ADSM clients?

The results illustrated in Table 46 on page 185, Table 47 on page 185, Table 48 on page 186, and Table 49 on page 186 show that, for large files, there is little benefit in running the ADSM client on more than one SP node.

For small files, a worthwhile performance improvement was measured when running the ADSM client on two and four SP nodes.

Since there was no significant disadvantage for large files in using multiple SP nodes for the ADSM client, you should base your decision on the volume of small files that you need to back up.

There was no benefit in running the ADSM client and server on the same SP node, and resource constraint affected the overall throughput for large files.

Chapter 6. Test results

This chapter contains the results of all the tests. The tests outlined were created out of the desire to answer the following questions:

1. How does the choice of file system type (JBOD), replicated, mirrored, or RAID-5 affect performance?
2. How does the number of SSA loops and disks on the loop affect performance of GPFS?
3. How does the applications I/O access method affect performance?
4. For RAID-5 file systems, how does the choice of 4+P, 7+P, and 15+P array sizes affect performance?
5. By how much will the performance of a GPFS file system using a RAID-5 disk change by putting the metadata on a non-RAID disk?
6. Should the VSD Servers be dedicated, or can they be used to run other jobs?
7. How many VSD servers is it best to have for a given GPFS file system?
8. How does GPFS performance vary with VSD server node type?
9. What are the constraints on GPFS client node data throughput?

The *Base Run tests* were constructed to cover a lot of these questions using a stable set of hardware that could be easily configured.

RAID-5 Array size tests and *Metadata tests* were also performed in an attempt to cover those remaining unanswered questions

6.1 Base run tests

For the Base Runs, the following types of test are carried out on a number of different GPFS file system configurations.

- Serial
- Parallel
- Random

Each Run usually covers all the above mentioned tests on a particular configuration.

6.1.1 Serial tests

Serial tests are comprised of writing and reading a large file to the GPFS file system. This was done from 1 then,2 and then 4 client nodes simultaneously. The application on each GPFS client node writes and reads to a separate file at the same time. The application block size was always 256KB and the file size was at least 1 MB. All tests utilized the iopm program described later.

6.1.2 Parallel tests

Parallel tests were comprised of writing and reading a single 16 GB file to the GPFS file system. This was done from one, then two, and then four client nodes simultaneously. The applications on each GPFS client node writes and reads to the same file at the same time. The application block size was always 256 KB, and the data partitioning file layout was always segmented, which means that in the 4 client case, for example, each node reads and writes to a continuous 4 MB section of the file. All tests utilized the ior program described later.

6.1.3 Random tests

Random tests were comprised of writing and reading a 512 MB file to the GPFS file system. This was done from one, then two, and then four client nodes simultaneously. The applications on each GPFS client node writes and reads to the same file at the same time. The application block size varied between 400 and 40,000 bytes (equivalent to 100 integers to 10,000 integers). The offset into the file was varied as well. All tests utilized the ior program described later. These test were only performed on the S1.C4 test environment.

6.1.4 Configurations

Hardware - For the Base Runs, we had a frame of eight 332 MHz SMP wide nodes each with 3 GB of memory two IBM Enhanced RAID adapter cards(feat:6215). We had eight SSA drawers of 4.5 GB Scorefire disks available.

Software - Each node was loaded with the following software:

- AIX 4.3.2
- PSSP 3.1.0.6
- RVSD 3.1.0.3
- VSD 3.1.0.4
- GPFS 1.2.0.3

GPFS setup

There were either two or four dedicated VSD Server Nodes used. There were four nodes dedicated for GPFS Client Nodes. GPFS file system block size was kept at 256 KB and all GPFS parameters were as recommended in installation section of the *GPFS Installation and Administration Guide*, SA22-7278, apart from the following:

- Buddy Buffers - 33 for VSD servers and 2 for GPFS Nodes

Switch parameters were set to the following on all nodes

- thewall - 65536
- sb_max = 1310720
- tcp_sendspace = 327680
- tcp_recvspace = 327680
- udp_sendspace = 65536
- udp_recvspace = 655360
- rfc1323 = 1
- tcp_mssdflt = 1448
- ipforwarding = 1

For the Basic Runs, we ran tests on the test environment described by three parameters: Setups, Configurations, and Tests as shown in Tables 28 through 30.

Table 28. Test environment — Setups

Setups	Description
S1	defines 8 disks per SSA loop.
S2	defines 16 disks per SSA loop

Table 29. Test environment — Configurations

Configurations	Description per VSD Server Node
C1	1 SSA loops/adapter, 2 SSA Adapter cards
C2	1 SSA loop/adapter, 2 SSA Adapter cards
C3	2 SSA loops/adapter, 2 SSA Adapter cards
C4	1 SSA loop/adapter, 1 SSA Adapter card

Table 30. Test environment — Tests

Tests	Description
T1	2 VSD Server nodes, JBOD file system
T2	4 VSD Server nodes, JBOD file system
T3	4 VSD Server nodes, RAID-5 file system
T4	4 VSD Server nodes, LVM Mirrored file system
T5	4 VSD Server nodes, GPFS Replicated file system

This enables *s1.c4.t5* to define the configuration for a particular run of tests.

The s1.c4.t5, for example, describes Setup 1 (eight disks per SSA loop), configuration 4 (one adapter card with one SSA loop), and test 5 (using four VSD server nodes and the GPFS file system built with replication across the four VSD servers).

The SSA drawers were wired symmetrically between pairs of nodes. To give eight disks per loop, layout 1 was used such that an SSA drawers was wired between a pair of adapter cards in the following node pairs: 1-3, 5-7, 9-11, 13-15. Each node has two SSA adapter cards, therefore, using all eight drawers.

To give 16 disks per loop, the wiring was then changed to layout 2 giving two SSA drawers wired between a pair of adapter cards in the following node pairs: 1-3, 5-7.

For setup 1, with configurations 4 and configuration 2, we used nodes 1,3,5,and 7 as the VSD server nodes using the layout 1 disk configuration.

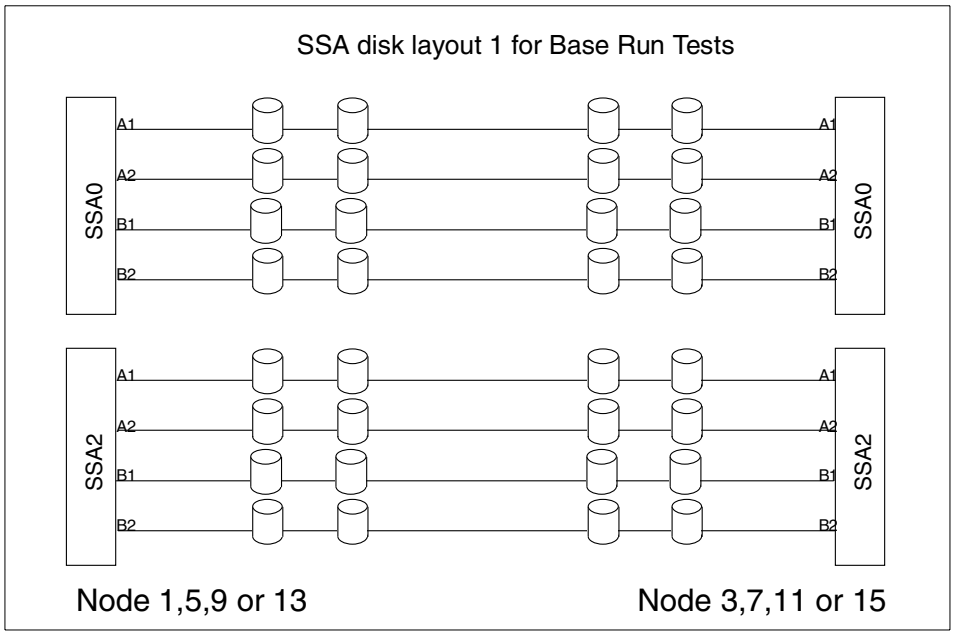


Figure 60. SSA disk layout 1 — One SSA drawer per node

For setup 1, with configuration 3, we had to use nodes 1,5,9 and 13 so that we could use all the disks connected to those nodes.

For setup 2, with configuration 1, we used nodes 1,3,5 and 7 with the disks rewired in to layout 2 format.

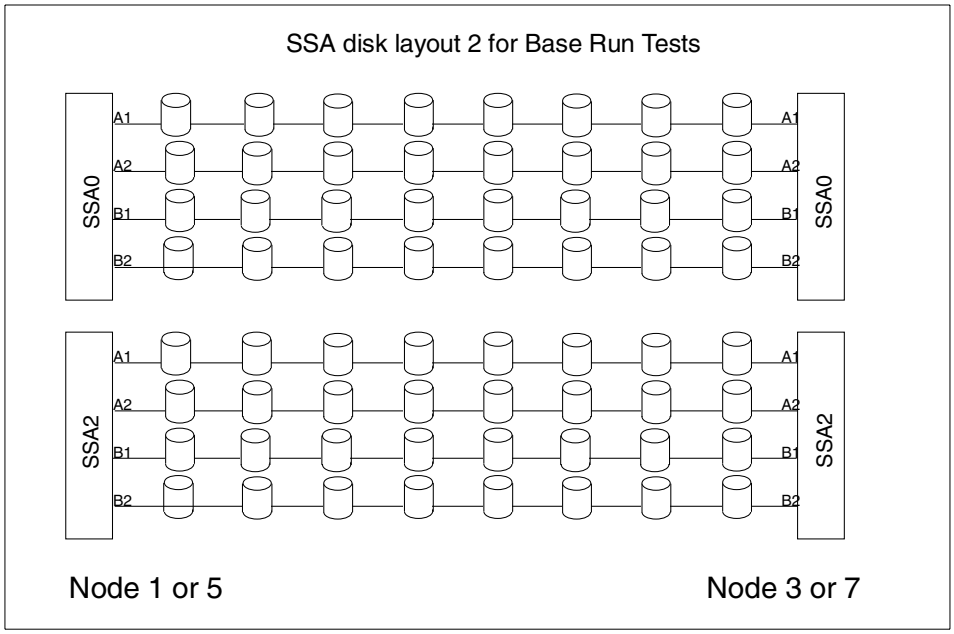


Figure 61. SSA disk layout 2 — Two SSA drawers per node

6.1.5 Applications

In collecting our tests results, we used the following applications:

- **ior.** This application performs parallel reads or writes to the same file from multiple nodes. It has data partitioning capabilities for mapping the parallel tasks to a single global byte stream in a round-robin file layout or a segmented file layout. Furthermore, it is able to write random xfer sizes to random locations in the file. The application logic is: (1) barrier; (2) collect start time stamps; (3) open file; (4) perform multiple reads (or writes) until the entire file is processed; (5) close file; (6) collect end time stamps; (7) compute elapsed time as latest end time stamp minus earliest start time stamp; (8) report MB/sec; (9) sleep for 15 seconds and repeat. Note that the times for open and close are included in calculating bandwidth results. All results are reported as aggregate rates. It is an MPI application written in the C language. The author for ior is Robert Kim Yates.
- **iopm**. This application performs one or more serial reads or writes. It has the capability of writing and/or reading multiple files, each as an independent serial process, and reporting the aggregate performance. When performing tests with multiple files, children processes are created via the fork system call. The application logic is: (1) open separate file for

each write or read process; (2) collect start time stamp; (3) if more than one reader or writer, fork processes for read or write; (4) each process performs read or write; (5) each process closes file; (6) wait for processes to complete; (7) collect end time; (8) calculate elapsed time and report aggregate results. Unlike ior, iopm does not include the time to perform file opens in the bandwidth results. It is written in the C language.

- **ADSM application**: ADSM is capable of reporting its performance. This feature was employed for all ADSM numbers.

6.1.6 Measurement tools

The measurement tools used in this section are all from the *Performance Toolbox Version 1.2 and 2 for AIX: Guide and Reference,* SC23-2625

These tools were used to monitor the activity on all nodes. The activity information used in the analysis is:

- CPU information
- IP packet drop information on VSD server nodes
- VSD retries up to level 3 for GPFS client nodes

On each node in our system, a daemon called xservd is running and collects statistics about that node every 30 seconds. This data is saved in the /etc/perf directory on that node in a file named azizo.<date>. The variables that are monitored are set up in the file /etc/perf/xmservd.cf. This is the same for all nodes.

At the end of each separate test, these files on each node are copied to the control workstation to be saved, and the name is changed to incorporate the node.

These files now contain the measurements taken during each test, and the following tools can be used to filter the information for graphing.

- ptxtab — Used to create tables that are acceptable for spreadsheet analysis.
- ptxsplit — Used to split the recording file into multiple files containing particular variable observation sets.
- ptxmerge — Used to recombine split files into one that perhaps contains multiple node information for a particular variable observation set.

6.1.7 Measurements

The measurements taken are set out in Appendix A, “Measurements” on page 225. Measurements are split into Serial, Parallel, and Random test sections. Within each section, each graph has a label (Server View - s1.c4.t3.s) that confirms the view, either Client or Server, the setup, configuration, and test environment.

The data has been filtered to give both a Client view, which is the average of all the GPFS nodes operating, and also a Server view, which is the average of all the VSD Servers.

The information displayed by the graphs is the data throughput rates for both read and write operations along with the CPU usage.

The Read or Write CPU percentage given is the addition of User and System time. The Read or Write CPU Wait percentage given is just the Wait time. Client Views do not have Wait time because they are not using local disks, and this, therefore, is not considered relevant.

6.2 RAID-5 array size tests

The purpose of these tests was to investigate what the effect of varying the RAID-5 arrays’ size has on performance. For this test, the hardware and software setup was the same as for the Base Run tests.

The tests for 4+P RAID-5 array size was done as part of the Base Runs and was done for all the various test environments.

Here, we chose setup 2, configuration 1, and performed the same tests as for the Base Run tests on s2.c1 configuration with only the size of the RAID-5 array changed to 7+P and then 15+P.

The parallel tests were used as described in Section 6.1.2 “Parallel tests” on page 200

6.2.1 Measurements

The measurements taken are set out in A.2, “RAID-5 array size tests” on page 271.

6.3 Metadata tests

The purpose of these tests was to investigate the effect of a metadata intensive application with GPFS. We also want to see, for a GPFS RAID-5 file system, the difference in performance between having the metadata on the RAID-5 arrays versus having it separated on a separate JBOD disk.

For this test, the hardware and software setup was the same as for the Base Run tests.

The application used is the SPEC SDM Version 1.1 application running the 057.SDET Benchmark. This benchmark is sponsored by the AT&T Computer Systems Division of the AT&T Bell laboratories. It consists of a number of scripts running concurrently that each emulate various typical AIX commands being run by multiple users. For example:

- Changing directories - cd
- Editing a file - ed
- Moving a file - mv
- Copying a file - cp
- Removing a file - rm
- Listing a directory - ls

Different multiples of scripts were run on each node to simulate varying numbers of user activity. The benchmark application was run on the following GPFS file system setups:

- Setup 2 configuration 1, 3 x 4+P RAID-5 arrays per server. Both data and metadata combined.
- Setup 2 configuration 1, 3 x 4+P RAID-5 arrays per server with data only. One JBOD disks per server for the metadata.
- Setup 2 configuration 1, JBOD file system with 16 disks per server all with data and metadata.
- Setup 2 configuration 1, JBOD file system with 16 disks per server. 15 with data and 1 with metadata.

Some of these tests were carried out on four GPFS client nodes concurrently active, and some were done on only one GPFS client node.

6.3.1 Measurements

The measurements taken are set out in A.3, “Metadata tests” on page 273.

6.4 Client max throughput tests

The purpose of these tests was to investigate why, in the base run tests, the maximum throughput a client sees is about 50MB/sec.

The Serial test were used as described in Section 6.1.1 “Serial tests” on page 200. One GPFS client node was used, and the application was configured to be able to run multiple threads on this node.

The test environment chosen for this test was setup 2, configuration1, test 2, which is a 4 VSD Server node configuration running a JBOD file system. This was chosen to have the least likelihood of experiencing server bandwidth constraints.

6.4.1 Measurements

The measurements taken are set out in A.4, “Client max throughput tests” on page 275.

6.5 Analysis

In this section, we attempt to answer most of the questions posed at the start of this chapter. The analysis uses the test results in Appendix A, “Measurements” on page 225 to demonstrate the reasoning behind the answer presented.

6.5.1 Compare RAID-5, mirroring, replication, and JBOD

Both Figure 62 on page 209 and Figure 63 on page 210 show the performance of a 4 VSD server GPFS configuration that uses setup 1, configuration 3, and four clients. This has the fewest bottlenecks identifiable using dropped packets. Only the Replicated file system displays any dropped packets.

For writes, the JBOD file system records the highest data throughput followed by RAID-5, mirrored, and then replicated. Even with replicated being the slowest, it was still the only one to cause IP packets to be dropped. This is probably due to the increased switch traffic caused by GPFS writing the data twice.

For both the mirrored and replicated case, it is worth pointing out the bandwidth required of the SSA adapter and the disks is twice that for JBOD for the same client load. The bandwidth required of the switch in the replicated case is double that for JBOD; whereas, for the mirrored case, it is

the same as for JBOD, as it is only when the client data reaches the VSD servers that AIX steps in and does the mirroring.

For read, all the file systems have very similar performance, with JBOD and Mirroring being slightly faster.

It is notable that, from the clients view, the MB/sec written per CPU cycle is about one for all but the replicated file system whose value is about half that.

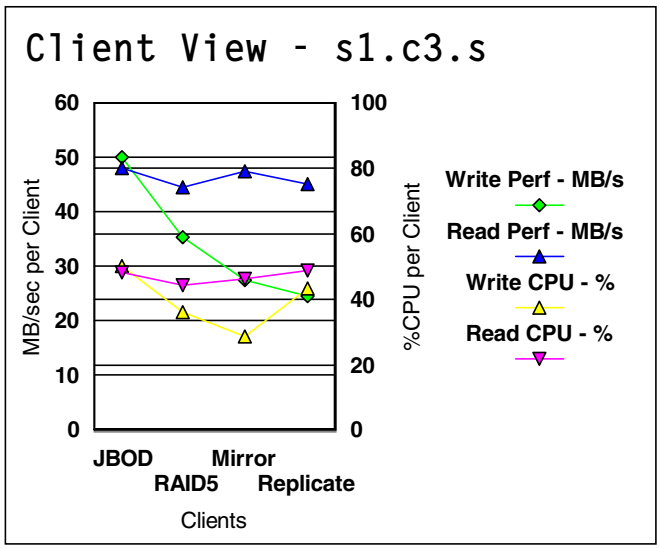


Figure 62. Client comparison: GPFS file system types — 4 VSD / 4 GPFS nodes

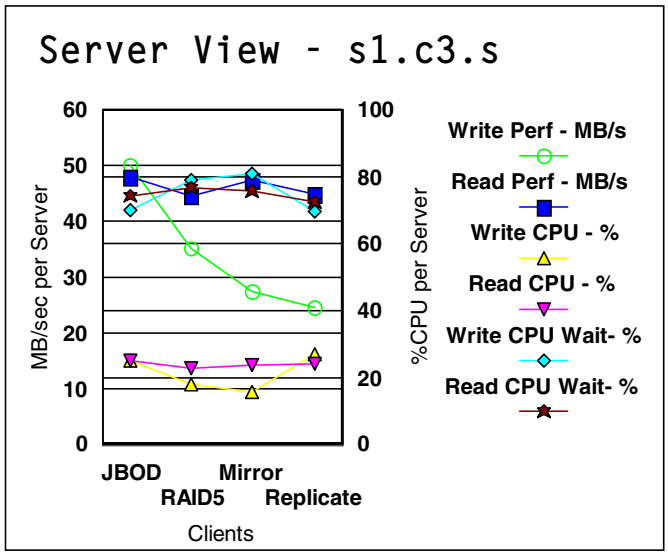


Figure 63. Server comparison: GPFS file system types — 4 VSD / 4 GPFS nodes

6.5.2 Compare SSA disk and loop combinations

First, we compare how varying the number of SSA loops affects performance. We have fixed on eight disks per loop, four GPFS client nodes, and two VSD servers running with a JBOD file system. This equates to Setup 1 Test1.3.s.

Figure 64, “Server performance with 4 GPFS clients and 2 Servers JBOD” on page 211, shows that, for two VSD servers, increasing the number of loops from two to four for each VSD server has no effect on overall performance. This is probably because the data throughput per server is being limited by the SSA adapter cards. From Table 8 on page 91, we see that, theoretically, two 6215 adapter cards can supply 43.5 MB/sec write and 59 MB/sec read. In practice, we recorded slightly higher figures for the write and slightly lower figures for the read.

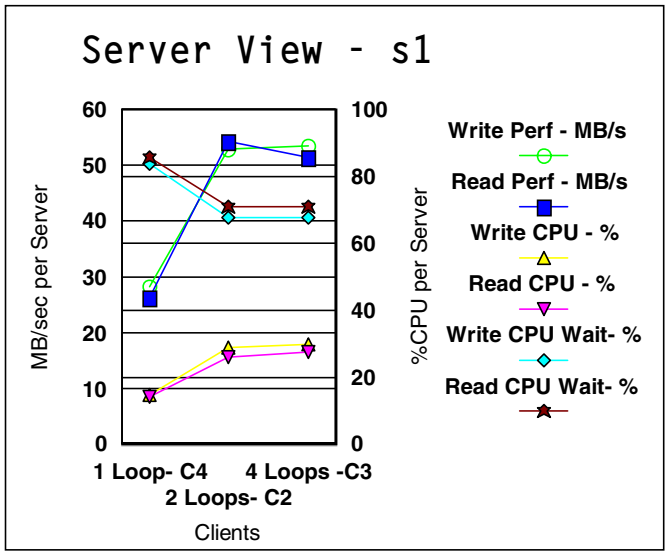


Figure 64. Server performance with 4 GPFS clients and 2 Servers JBOD

Now, we examine how varying the number of disks per loop affects performance. For this, we have fixed on one loop per adapter, four GPFS client nodes, and two VSD servers running a JBOD file system.

Figure 65, “Server performance: 1 loop per adapter — 4 clients / 2 Servers JBOD” on page 212, shows the same throughput increase from eight to 16 disks as Figure 64. This is because they are the same data points.

The main difference between Figure 65 and Figure 64 is for the 32 disk case. For the SSA disks graph, the 32 disks are spread across two SSA loops, one on each of two SSA adapters; whereas, for the SSA loops graph, the 32 disks are spread across the four available loops of the two SSA adapter cards

We can see that the maximum data throughput rate in both graphs is limited by adapter cards and not disks. The fall in the write throughput at 32 disks is probably caused by the extra contention of the 16 disks per SSA loop as opposed to having eight disks per loop.

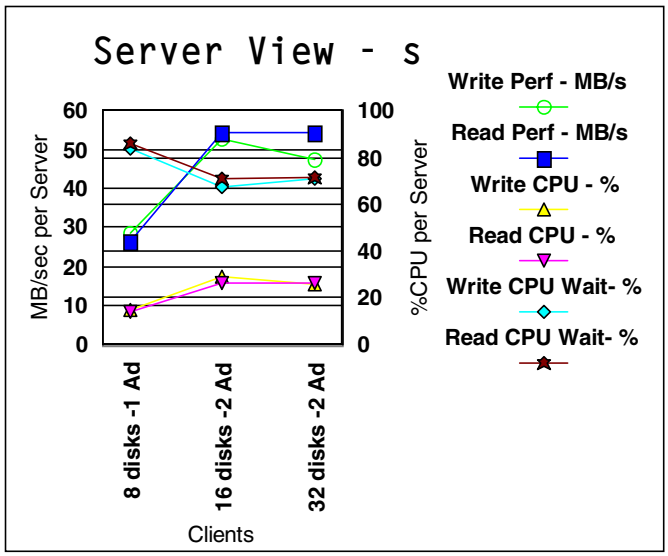


Figure 65. Server performance: 1 loop per adapter — 4 clients / 2 Servers JBOD

6.5.3 Compare serial parallel and random application performance

Comparing the Serial and Parallel tests for any particular test environment shows that the performance figures correspond very closely.

For example, let us compare using the Client view for test 3, setup1, configuration 4. Comparing Figure 66 and Figure 67, we can see this.

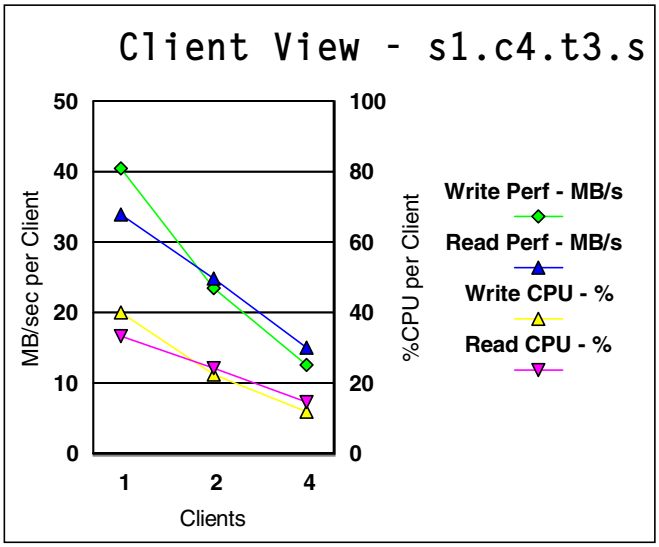


Figure 66. Performance graph s1.c4.t3 — Client View: Sequential

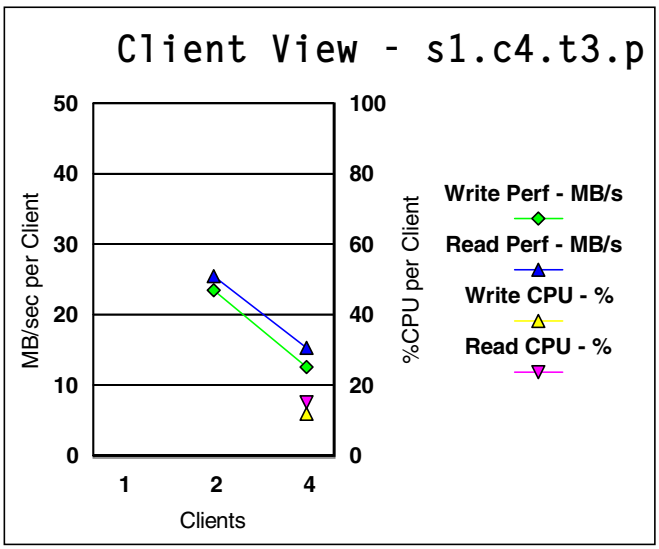


Figure 67. Performance graph s1.c4.t3 — Client View: Parallel

From Figure 68, we can see that the performance of random I/O is far below that for Serial or Parallel tests. While the read and write performance for sequential access for two clients is about 25 MB/sec the performance when using random access is shown to be 2 MB/sec for read and less than 1 MB/sec for write.

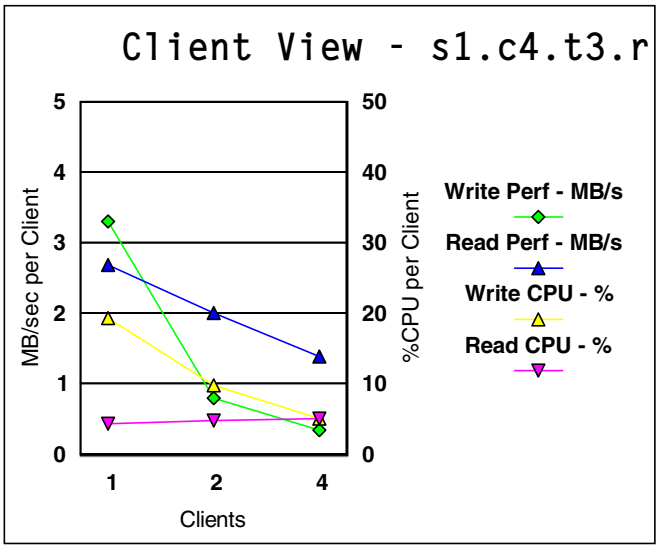


Figure 68. Performance graph s1.c4.t3 — Client View: Random

6.5.4 Compare RAID-5 with different array sizes

From Figure 69, we can see that the write performance is much better for the 4+P arrays size. This is because, in this case, the application is writing with 256 KB block sizes to the GPFS file system, which is created with a 256 KB block size. This means that the write is a strided write such that it can be done in one operation to all 4+P disks. For the other array sizes, the write is non-strided, and because the block being written does not fit exactly into the four 64 KB strides on each of the data disks, after the first write of the 256 KB data block, the complete stride has to be read, parity calculated, and the entire stride written back. In this instance, we can see that the write performance for a non-strided write is 38 percent of that on the strided write.

The read does not suffer from the same problem, and the read performance only shows a small decrease from 7+P to the 15+P case. This is probably due to I/O contention of the 16 disks on one loop.

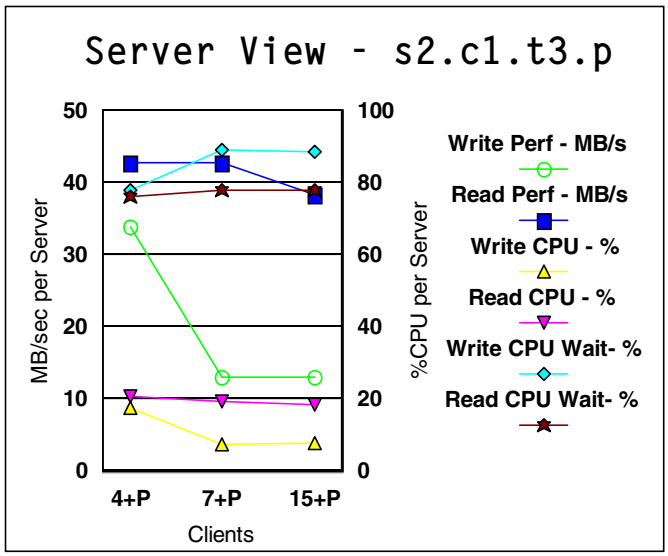


Figure 69. RAID-5 server view performance with different arrays sizes — s2.c1.t3

6.5.5 Compare RAID-5 with/without metadata on RAID-5

Comparing Figure 163 on page 273 with Figure 67 on page 213, we can easily see that, in this case, where a metadata intensive application is being run concurrently on a four GPFS nodes, that the performance is generally higher for the case when the metadata is combined with data on the RAID-5 arrays.

This is due to the fact that there are 12 data disks per VSD server over which to spread the metadata when it is combined with the data, and only one JBOD disk is used for the metadata when it is separated from the data.

So although writing and reading from a single RAID-5 disk is much slower than for a JBOD disk, in this case, this is more than compensated for by gains in performance caused by having 12 disks to use for metadata when on RAID-5.

6.5.6 Investigation of maximum client data throughput

From A.1.1.17, “S2_C1_Tests2” on page 242, we can see two interesting points:

1. The performance of a single node increases until we have four application threads operating on that node.
2. The write performance is higher than the read.

Point 1 is explained by the fact that the GPFS client node, in this case, has four processors and that each thread is constrained by the memory bandwidth of the node type.

Point 2 is explained by the fact that the read performance of the client node is constrained by that fact that it is driven by interrupt processing waiting on packets arriving and further hampered by locks that occur on the read threads. The write performance is not constrained by the same interrupt driven process and is only constrained by the bandwidth of the memory movement of the node.

6.5.7 Analysis of CPU usage with regard to dedicated VSD servers

In an attempt to address the question of whether or not to have dedicated Servers, let us first look at the CPU usage profile from the tests carried out. We will then be able to consider how to calculate the spare CPU capacity that the VSD servers may have available to dedicate to other tasks.

6.5.7.1 Clients (application nodes)

By inspection of the Client View performance graphs in the Serial section of A.1, “Base runs” on page 225, it can be seen that the ratio of CPU percentage to MB/sec data throughput for both read and writes is about 1:1. That is to say that, for a GPFS client node to write and read, 1 MB/sec sequentially consumes about 1 percent of CPU. The only exception to this is for the replicated file system where the write takes twice this; so, the ratio of CPU percentage to MB/sec written in the replicated case increases to about 1.8:1.

This makes sense because it is only in the in the replicated case that the client is responsible for issuing both the replicated writes.

We propose the following guidelines for estimating the maximum CPU utilization for GPFS client nodes where the application I/O patterns are large block and sequential:

- Begin by assuming 1 percent of CPU for each MB/sec from the client.
- For GPFS replication, assume 1.8 percent of CPU for each MB/sec written by the client.

By inspection of the Client View performance graphs in the Random section of A.1, “Base runs” on page 225, it can be seen that the ratio of CPU

percentage to MB/sec data throughput for writes is about 6:1. That is to say that, for a GPFS client node to write and read, 1MB/sec randomly consumes about 6 percent of CPU. The ratio of CPU percentage to MB/sec data throughput for read is about 1.8:1.The only exception to this is for the replicated file system where the write takes twice this; so, the ratio of CPU percentage to MB/sec written in the replicated case is 12:1.

Again, this is because it is only in the replicated case that the client is responsible for issuing both the replicated writes.

For random access, the limiting factor appears to be the single disk bandwidth for reads and writes larger than the GPFS blocksize (or around 3.82 to 7.42 MB/sec, see column four of Table 4 on page 80), and the single disk sector bandwidth for reads and writes is smaller than the GPFS blocksize (or around 0.2 MB/sec). We propose the following guidelines for estimating the maximum CPU utilization for GPFS client nodes where the application I/O patterns are small block (400 -40000 bytes) and random:

- Begin by assuming 6 percent of CPU for each MB/sec written by the client.
- Assume 1.8 percent of CPU for each MB/sec read by the client.
- For GPFS replication, assume 12 percent of CPU for each MB/sec written by the client.

6.5.7.2 Servers (VSD server nodes)

By inspection of the Server View performance graphs in the Serial section of A.1, “Base runs” on page 225, it can be seen that the ratio of CPU percentage to MB/sec data throughput for both read and writes is about 0.6:1. That is to say that, for a GPFS client node to write and read, 1MB/sec sequentially consumes about 0.6 percent of CPU. The only exception to this is for the replicated file system where the write takes twice this; so, the ratio of CPU percentage to MB/sec written in the replicated case is 1.1:1.

We propose the following guidelines for estimating the maximum CPU utilization for VSD Server nodes where the application I/O patterns are large block and sequential:

- Begin by assuming 0.6 percent of CPU for each MB/sec from the VSD Server.
- For GPFS replication assume 1.2 percent of CPU for each MB/sec from the VSD server.

By inspection of the Server View performance graphs in the Random section of A.1, “Base runs” on page 225, it can be seen that the ratio of CPU percentage to MB/sec data throughput for writes is about 10:1. That is to say

that, for a GPFS client node to write, 1MB/sec randomly consumes about 10 percent of CPU. The ratio of CPU percentage to MB/sec data throughput for read is about 3:1.The only exception to this is for the replicated file system where the write takes only slightly more than the normal write ratio CPU percentage to MB/sec of 12:1.

We propose the following guidelines for estimating the maximum CPU utilization for VSD Server nodes where the application I/O patterns are small block (400 -40000 bytes) and random:

- Begin by assuming 10 percent of CPU for each MB/sec written by the server.
- Assuming 1.8 percent of CPU for each MB/sec read by the server.
- For GPFS replication, assume 12 percent of CPU for each MB/sec written by the server.

6.5.7.3 To dedicate a server

If the requirements of a server are known, then, with the aid of the above figures, it is possible to estimate the likely CPU required of the server.

If this CPU figure is greater than 70 percent then is recommended that this VSD server should be dedicated to being a VSD server. To avoid the possibility of the GPFS file system becoming unavailable, it seems wise to only consider using the VSD Server node for other tasks if the expected GPFS related CPU usage is to be below 50 percent, and the other tasks are unlikely to cause the CPU to become 100 percent busy.

If the VSD server is expected to be less than 70 percent busy while serving GPFS, you may wish to consider placing other useful work on the VSD server. In such cases, a CPU-intensive task with little or no I/O and little or no switch communication is ideal. That is, the non-VSD workload should require minimal I/Os and minimal communication because these resources will be needed by VSD.

6.5.8 How the number of VSD servers affects performance

In general, performance is expected to linearly increase with additional VSD servers. This is what you would expect from a truly scalable file system. We, therefore, anticipate that a doubling in the number of servers would double write and read performance. This is observed in the *two-client* performance numbers for S1_C4_T1 and S1_C4_T2 in Figure 70 on page 219 (recall that the Sx_Cy_Tz nomenclature is defined in 6.1.4, “Configurations” on page 200). Four-client results would be preferable, but they were not available.

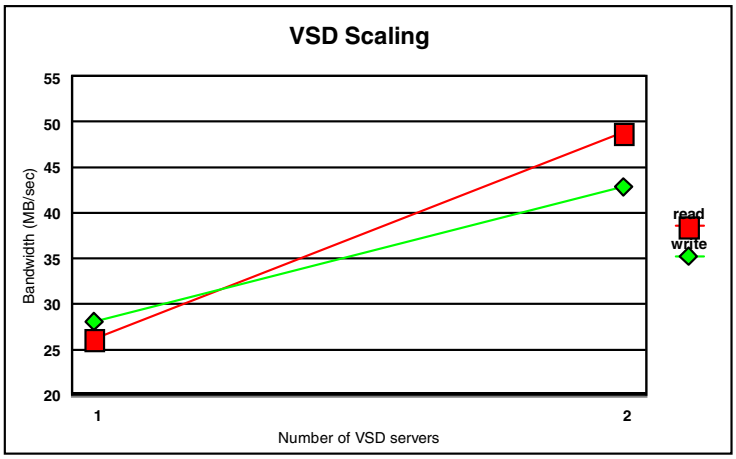


Figure 70. Results for two-client performance of S1_C4_T1 and S1_C4_T2

The following imbalances may negatively affect the GPFS scaling potential:

- Client constrained (too few clients): VSD server capacity goes untapped when there are not enough clients to drive them.
- Server constrained (too many clients): When clients are making requests faster than the VSD servers can process them, VSD server performance decreases. Depending upon the amount of demand, this decrease can be dramatic.

Unbalanced scenarios are shown in Figure 71 on page 220. On the left, we expect to double our performance when we add an additional VSD server going from the S1_C4_T1 configuration to the S1_C4_T2 configuration, but the results remain essentially constant due to client constraints. On the right, we expect to double our performance when we go from one client to four clients in configuration S1_C4_T1, but the results remain essentially constant because we are server constrained.

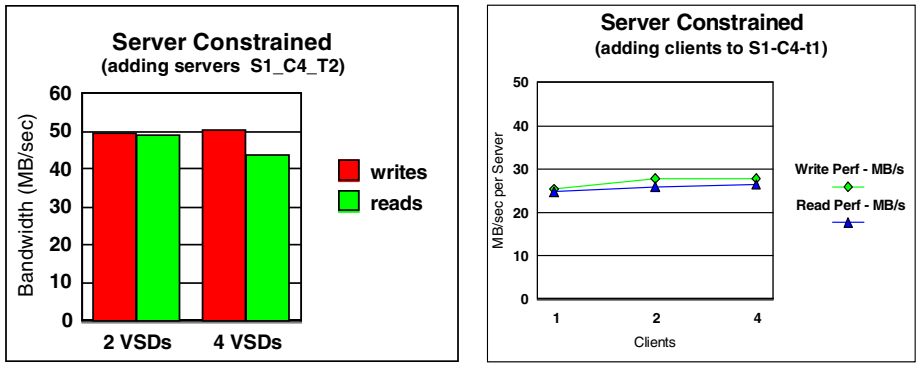


Figure 71. Server-constrained and client-constrained imbalances

6.5.9 Validating sizing

6.5.9.1 Sample 1: Setup S1_C4_T1

As an exercise in validating our sizing methodology, we construct an estimate of what each client is able to achieve. Then, we will perform a similar calculation of the aggregate VSD server bandwidth. We then compare our estimates with the actual performance recorded in A.1.1.1, “S1_C4_Tests1” on page 226.

By inspecting Table 28 on page 202 for the configuration of experiment S1_C4_T1, we have:

- Eight disks per SSA loop.
- One SSA loop and one Campbell SSA adapter per server.
- Two VSD servers with JBOD (no RAID, no mirroring) disks.
- Up to six Clients.
- All eight nodes are 332 Mhz 604e Silver nodes.

Following the guidelines set forth in Chapter 3, “Sizing GPFS” on page 71, we anticipate that the client bandwidth for a 332 Mhz SMP wide will be confined by the Switch throughput limit of around 76.3 MB/sec. We can also estimate the VSD server bandwidth using the following:

- Eight scorefire disks - 4.88 x 8 = 39 MB/sec max GPFS throughput (for disk performance, consult column four of Table 4 on page 80).
- SSA adapter card - 29.4 MB/sec read and 21.7 write max GPFS throughput (for SSA adapter performance, consult Table 8 on page 91).

- Each VSD Server should be able to provide a maximum GPFS throughput rate of about 30 MB/sec read and about 22 MB/sec write.

By looking at the Sequential Server view graph in Figure 73 on page 226, we see that using the 4 client case, the maximum measured figure for write was about 28 MB/sec and the maximum measured figure for read was about 26 MB/sec.

If we check the same measurements taken for the parallel test in Figure 73 on page 226, we also see very similar figures.

We see that the estimated throughput figures for the SSA adapter are perhaps too low for the write case and a little too high for the read case.

6.6 Conclusions

The following conclusions are for this chapter alone and are drawn entirely from the results of the experiments conducted.

6.6.1 File system options

Out of the file system options considered in this chapter, the JBOD file system provides the highest data throughput. If recoverability is also an issue, then the following file systems can be considered in their order of performance. Highest is given first.

1. RAID-5
2. Mirrored
3. Replicated

If using RAID-5, file systems use strided writes if at all possible. This requires 256 KB file system block size, 256 KB application block writes, and 4+P RAID-5 arrays. By comparison, non-strided writes are more than twice as slow.

6.6.2 SSA disk subsystems

To get access to the full bandwidth of SSA disks used, the following rules should be followed:

- Spread the disks evenly over all the available loops and adapter cards.
- Do not overload an SSA adapter card with too many disks. This turns out to be between eight to 12 disks depending on file system type, adapter card, and disk type.

6.6.3 Sequential I/O versus random

Sequential I/O for both read and write gives, by far, the best performance with GPFS file systems. By comparison, Random I/O is poor.

6.6.4 Metadata and RAID-5

Although putting metadata on RAID-5 arrays may seem a bad idea, as this will cause slow non-strided writes, in practise, it is better, in terms of performance, than using a smaller number of dedicated metadata JBOD disks.

6.6.5 Dedicated VSD servers

Although it may be possible for VSD servers to support other tasks, care should be taken that these tasks do not cause the CPU usage of the VSD node to become fully utilized.

6.6.6 General conclusions

- For best performance, GPFS requires site-specific tuning.

  Since GPFS is designed to be a general purpose file system, it must provide capabilities to be tuned to widely varying needs. In 4.1.2, “High impact issues” on page 103, we identify a number of high impact tunables that can have large impacts on important file system attributes, such as read and write performance. Our tests conducted for 4.3.1.3, “Running a parallel test on a replicated GPFS” on page 158 demonstrated a 34 percent improvement on random read rates, and tests conducted for 4.3.1.4, “Running a random test on a replicated GPFS” on page 162 demonstrated a 50 percent improvement on random writes. Depending on your sites needs, you may have to significantly change the default values. Therefore, some care must be given to adjusting GPFS for optimal performance.

- GPFS provides highest read/write bandwidth with sequential access patterns.

  Our results indicate that while sequential access patterns scale with the available hardware, random access patterns are limited to around 0.2 to 5.0 MB/sec. Parallel applications should use file layouts that provide sequential access for each node. In some cases, it may be beneficial to open a file multiple times to associate a sequential access with a particular file handle instance. See Chapter 2, “Application considerations” on page 57 for a discussion on GPFS programming techniques.

- GPFS is easy to use for application programmers.

The API for GPFS is POSIX. This makes GPFS easy to use for application programmers. None of the applications used to gather data for this book were modified for use with GPFS.

- GPFS provides scalable performance.

  Our results in 6.5.8, “How the number of VSD servers affects performance” on page 218 show that GPFS is able to utilize the available hardware. Furthermore, the section also demonstrates that GPFS is extensible should additional hardware become available.

- GPFS is built on an infrastructure of distributed services.

  The architecture of GPFS is discussed in Chapter 1, “GPFS architecture” on page 1. The GPFS daemon, mmfsd, runs on all nodes within a GPFS domain. Furthermore, GPFS depends on other subsystems, such as VSD servers, Group Services, the System Data Repository, and so on, which may also be distributed among any number of nodes. As a truly scalable system, GPFS distributes tasks among the available resources. Proper operation requires remote services running on nodes other than the application nodes. Our results demonstrate GPFS performance for the hardware at our disposal (parallel applications of up to four nodes and VSD server configurations with up to two servers).

- System administrators can expect a learning curve.

  During our results collection, we encountered several challenges in configuring and understanding the behavior our system. One such incident involved different classes of disk drives in the VSD storage devices. The slower drives caused unanticipated disparities between otherwise similar VSD servers. The cause was not immediately apparent.

Appendix A. Measurements

This appendix sets out the raw data for the following sets of tests.

- Base Runs
- RAID-5 Arrays size test
- Metadata tests
- Client max throughput tests.

A.1 Base runs

The base run test results are split into Serial Parallel and Random results sections.

Within each of these sections, there are subsections labelled, such as S1_C4_Test4, which, in order, signifies the Setup, Configuration, Test, all of which were described earlier. These subsections contain two graphs and some text. The graphs give both the Client view, giving the CPU and data throughput figures for the average single GPFS node, and the Server view, giving the CPU and data throughput figures for the average VSD Server node. The text gives additional information about the runs that is relevant. For example, if IP packet drops were detected for VSD server nodes, or if VSD retries were detected for GPFS nodes, then this information is given.

A.1.1 Serial

Presented below is the raw data from the Serial Base run tests using the iopm application.

A.1.1.1 S1_C4_Tests1

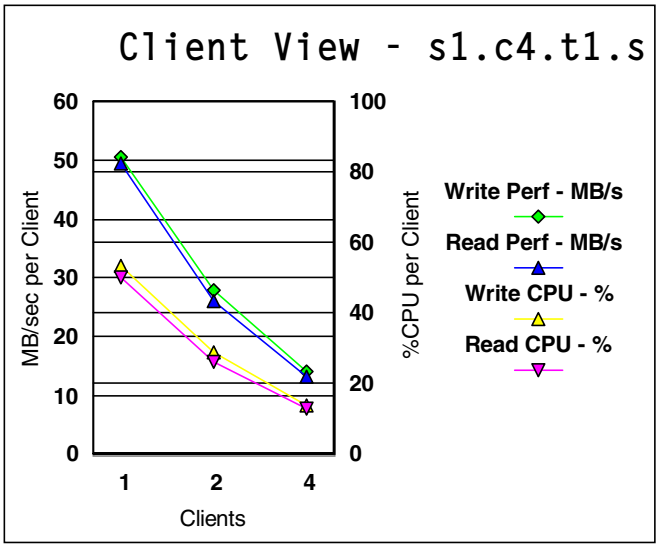


Figure 72. Performance graph s1.c4.t1 — Client view

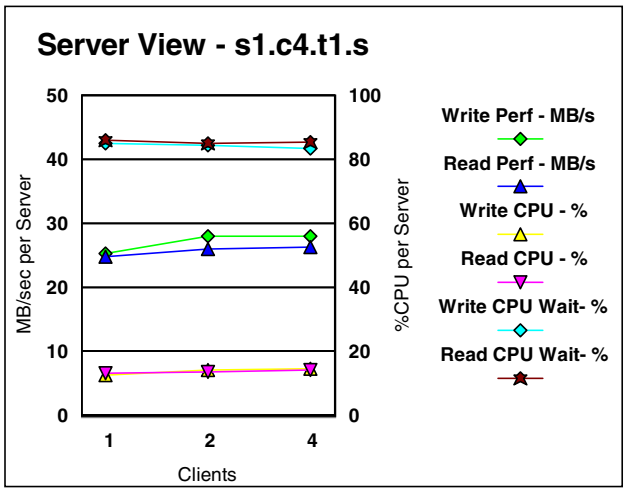


Figure 73. Performance graph s1.c4.t1 — Server view

A.1.1.2 S1_C4_Tests2

The data for four clients was not collected because the file system became unavailable. For the four clients run, server v06n07 was dropping IP packets causing VSD retries and eventually the application to fail.

Figure 74. Performance graph s1.c4.t2 — Client view

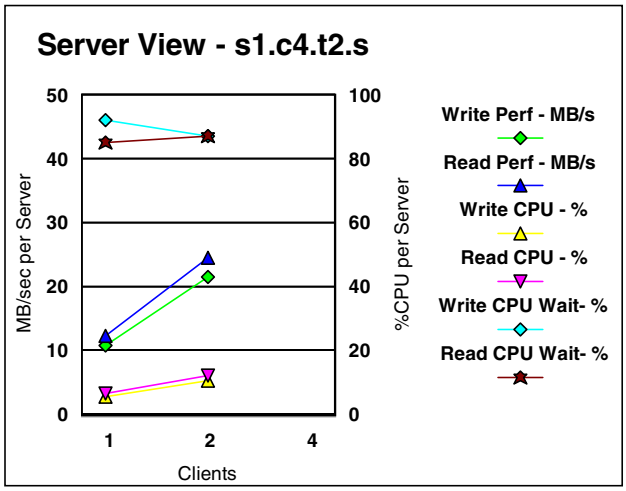


Figure 75. Performance graph s1.c4.t2 — Server view

A.1.1.3 S1_C4_Tests3

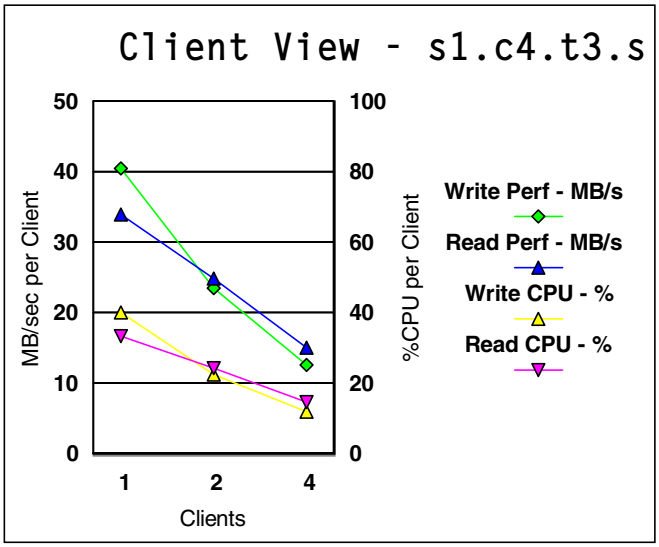


Figure 76. Performance graph s1.c4.t3 — Client view

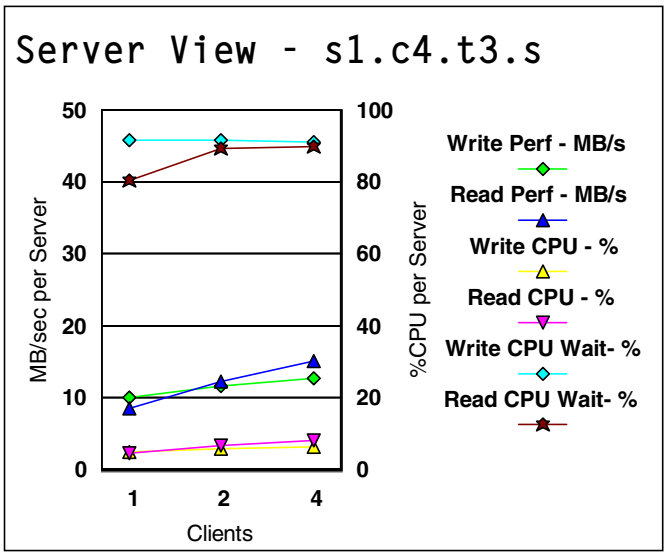


Figure 77. Performance graph s1.c4.t3 — Server view

A.1.1.4 S1_C4_Tests4

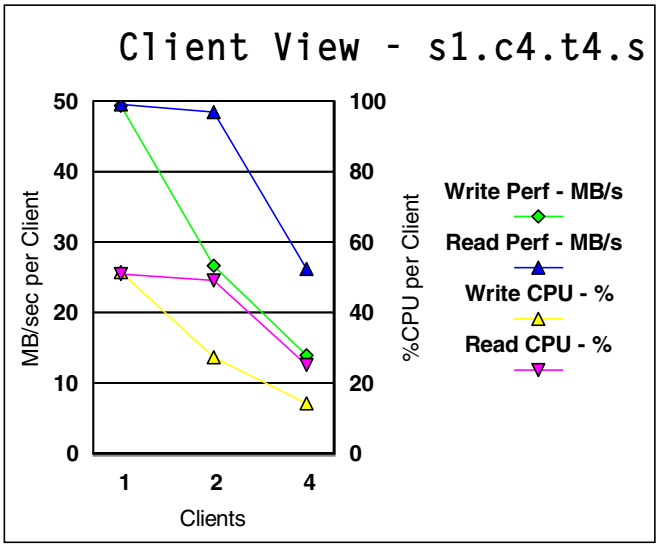


Figure 78. Performance graph s1.c4.t4 — Client view

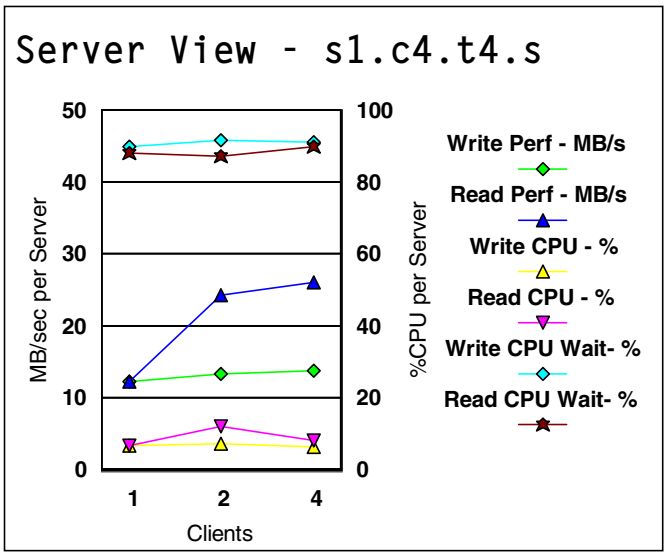


Figure 79. Performance graph s1.c4.t4 — Server view

A.1.1.5 S1_C4_Tests5

For the four clients run, the VSD servers are dropping IP packets causing VSD retries up to level 2, but the application does not fail.

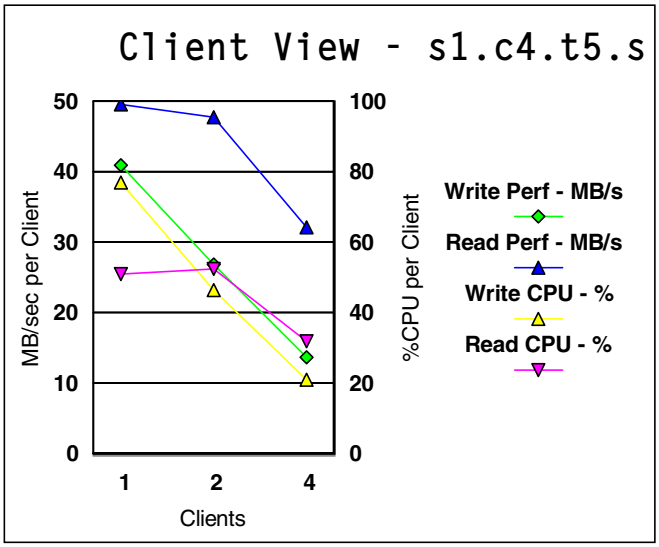


Figure 80. Performance graph s1.c4.t5 — Client view

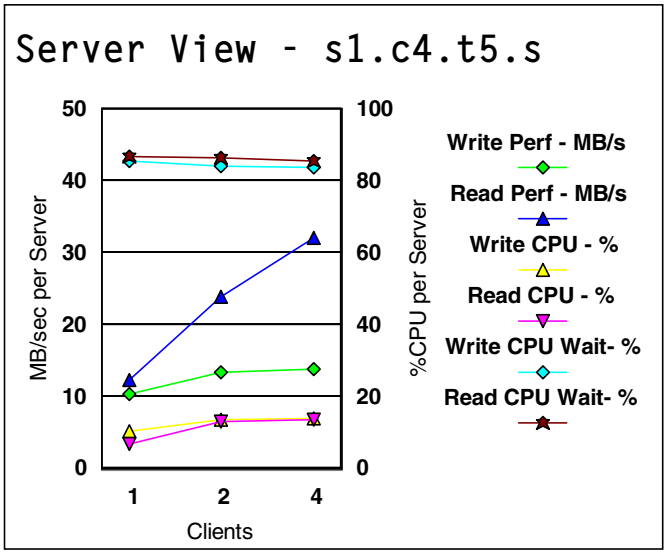


Figure 81. Performance graph s1.c4.t5 — Server view

A.1.1.6 S1_C2_Tests1

For the four clients run, VSD servers drop IP packets causing VSD retries up to level 2, but the application does not fail. Client v06n13 has one VSD retry level 3 recorded. The application does seem to temporarily stall during run4.

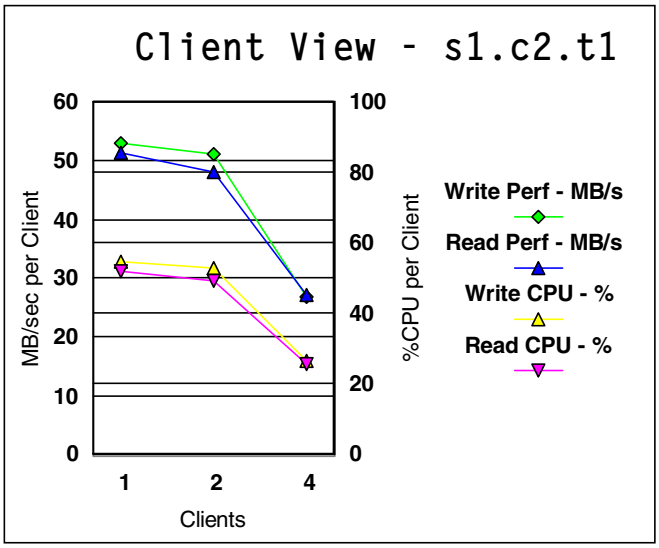


Figure 82. Performance graph s1.c2.t1 — Client view

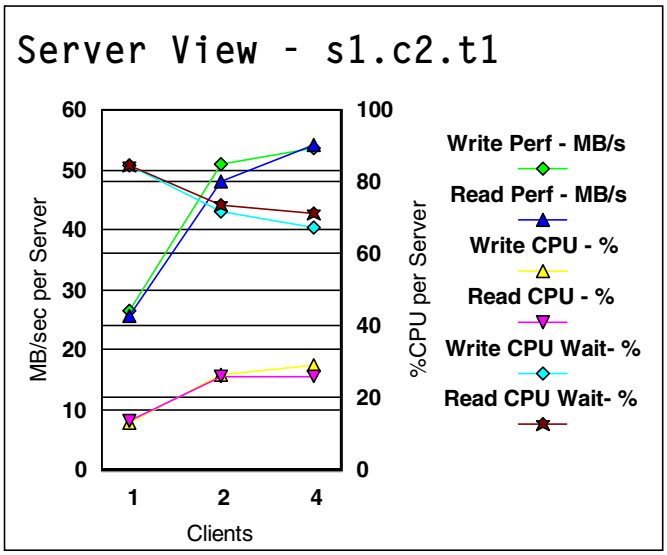


Figure 83. Performance graph s1.c2.t1 — Server view

A.1.1.7 S1_C2_Tests2

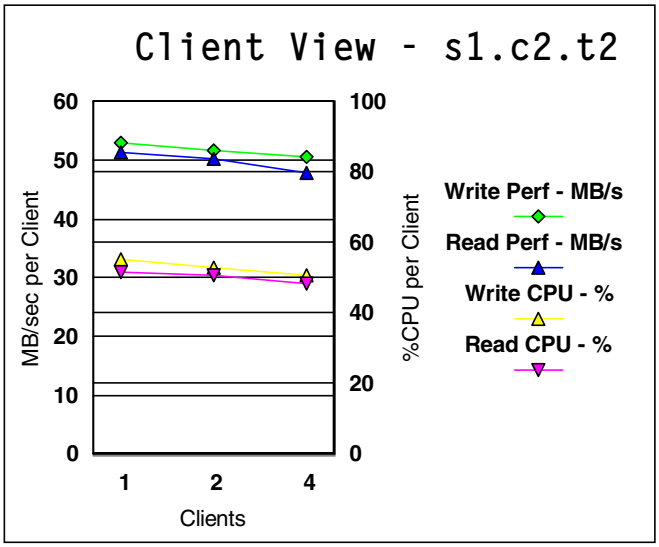


Figure 84. Performance graph s1.c2.t2 — Client view

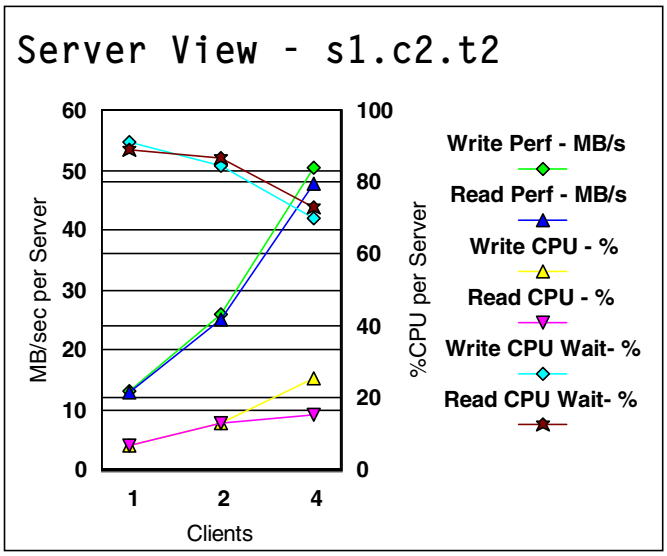


Figure 85. Performance graph s1.c2.t2 — Server view

A.1.1.8 S1_C2_Tests3

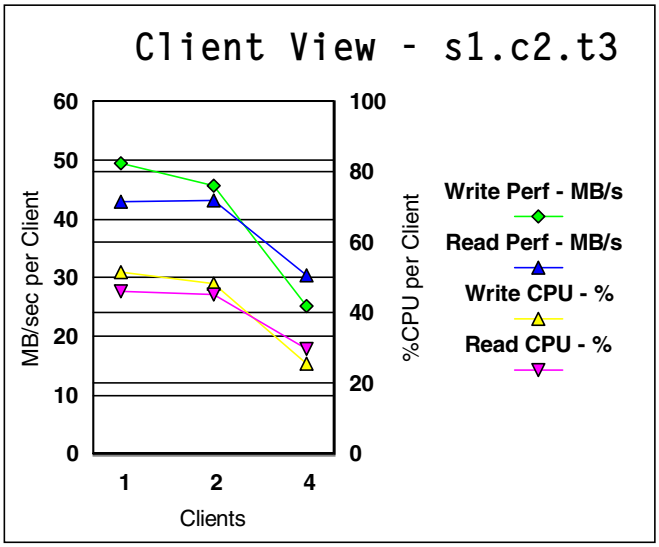


Figure 86. Performance graph s1.c2.t3 — Client view

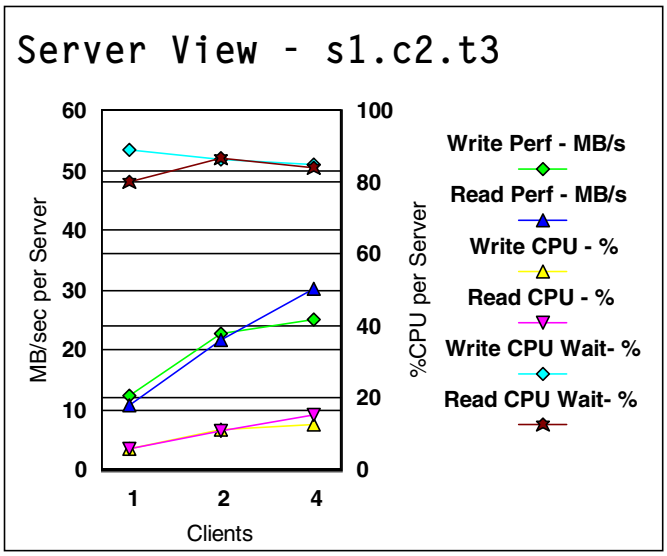


Figure 87. Performance graph s1.c2.t3 — Server view

A.1.1.9 S1_C2_Tests4

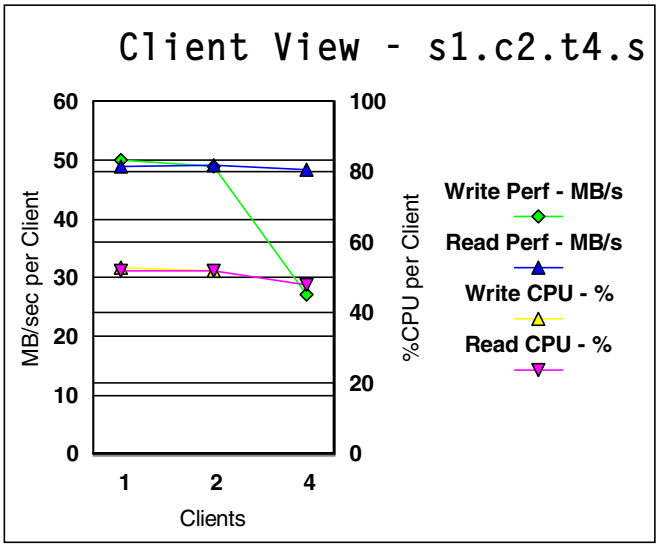


Figure 88. Performance graph s1.c2.t4 — Client view

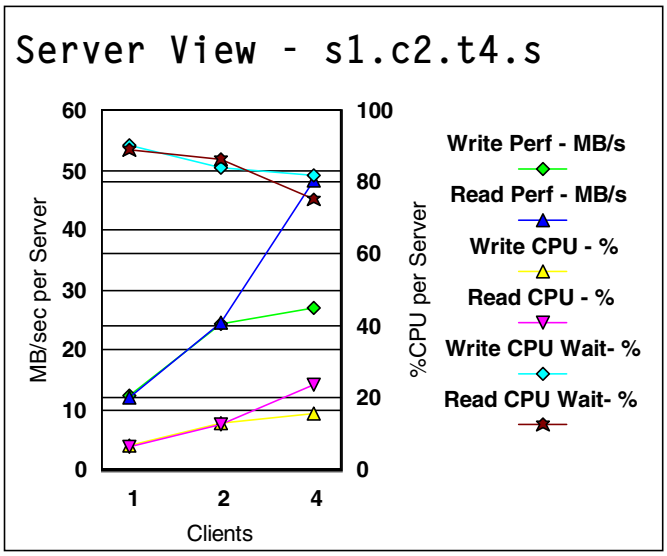


Figure 89. Performance graph s1.c2.t4 — Server view

A.1.1.10 S1_C2_Tests5

For the four clients run, the VSD servers are dropping IP packets causing VSD retries up to level 2, but the application does not fail.

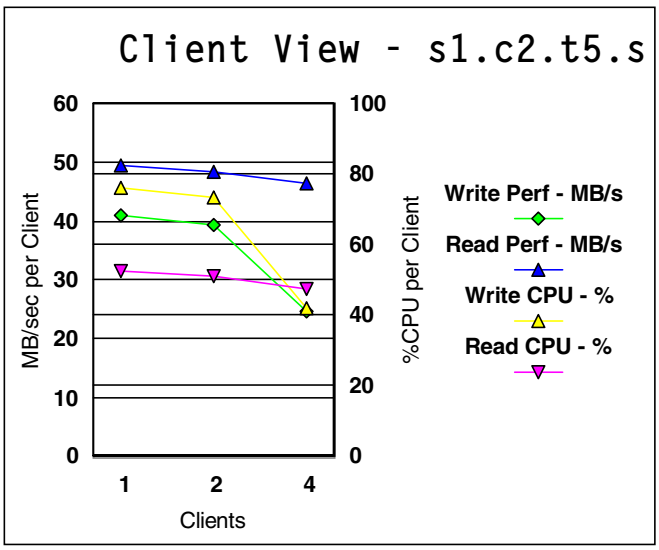


Figure 90. Performance graph s1.c2.t5 — Client view

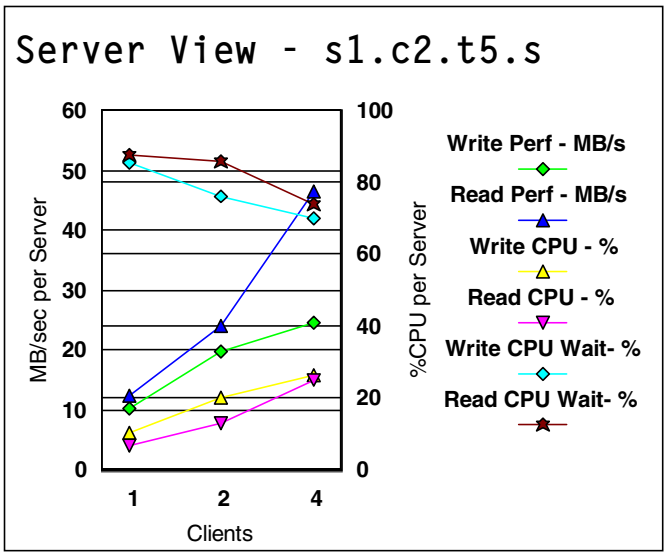


Figure 91. Performance graph s1.c2.t5 — Server view

A.1.1.11 S1_C3_Tests1

For the four clients tested, we got information for two runs. This may be because the file system became unavailable. Also, VSD servers dropped IP packets causing VSD retries up to level 2. The application fails at run 3.

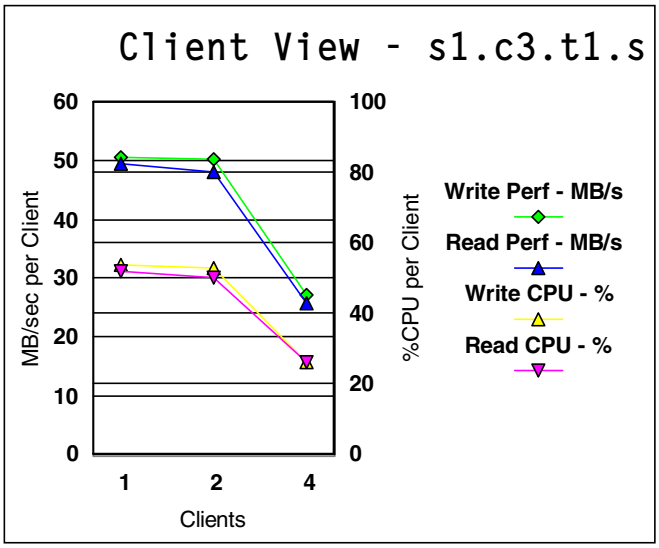


Figure 92. Performance graph s1.c3.t1 — Client view

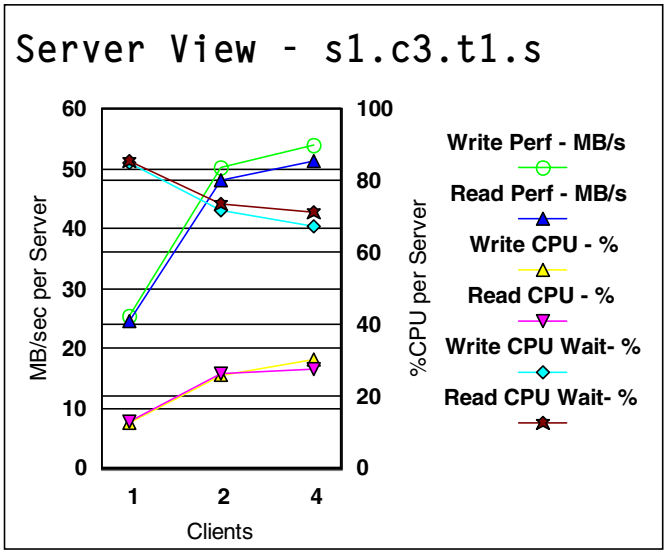


Figure 93. Performance graph s1.c3.t1 — Server view

A.1.1.12 S1_C3_Tests2

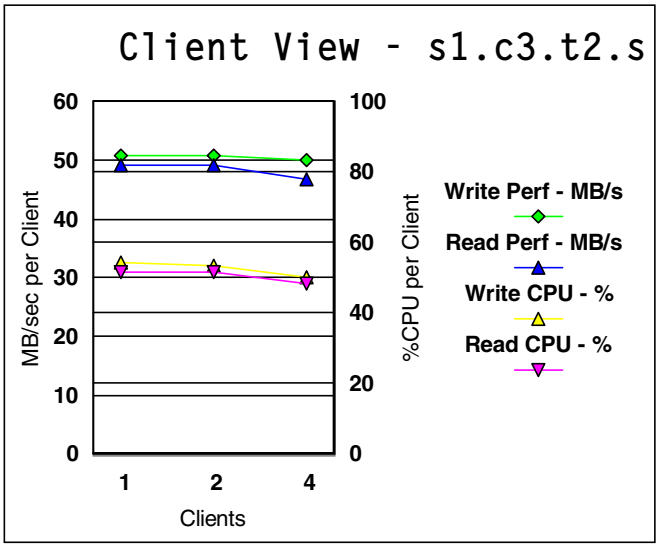


Figure 94. Performance graph s1.c3.t2 — Client view

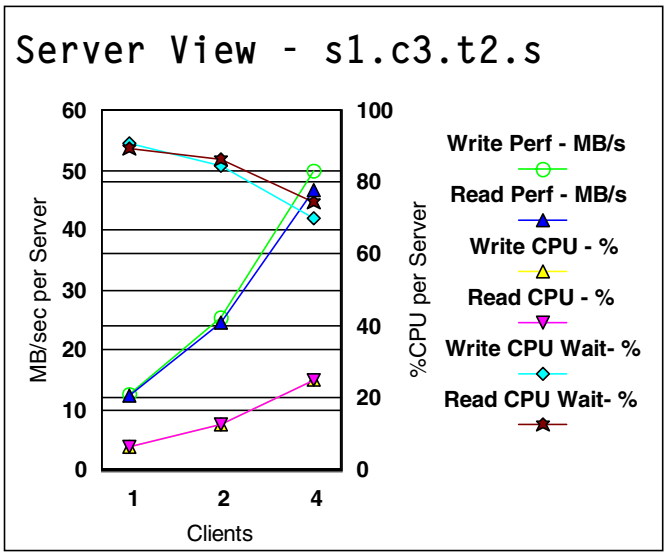


Figure 95. Performance graph s1.c3.t2 — Server view

A.1.1.13 S1_C3_Tests3

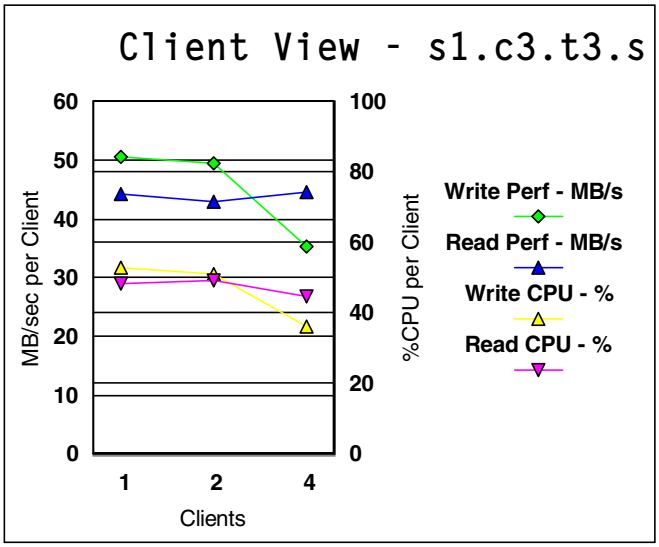


Figure 96. Performance graph s1.c3.t3 — Client view

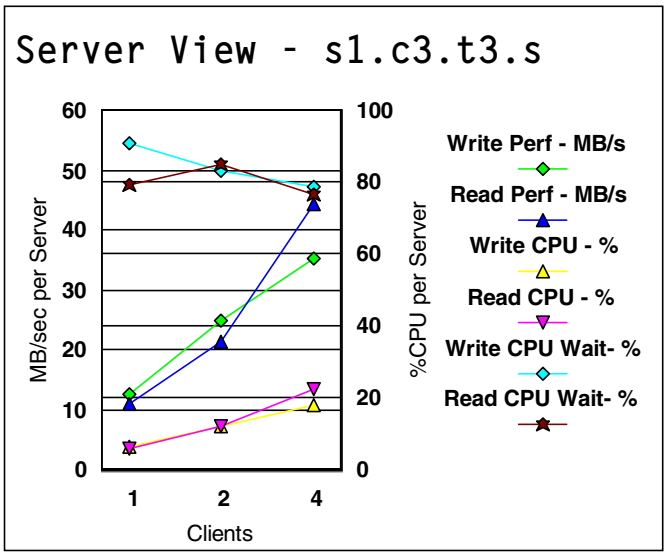


Figure 97. Performance graph s1.c3.t3 — Server view

A.1.1.14 S1_C3_Tests4

For the four clients run, the VSD servers are dropping IP packets causing VSD write retries on run 1 to go up to level 2, but the application does not fail.

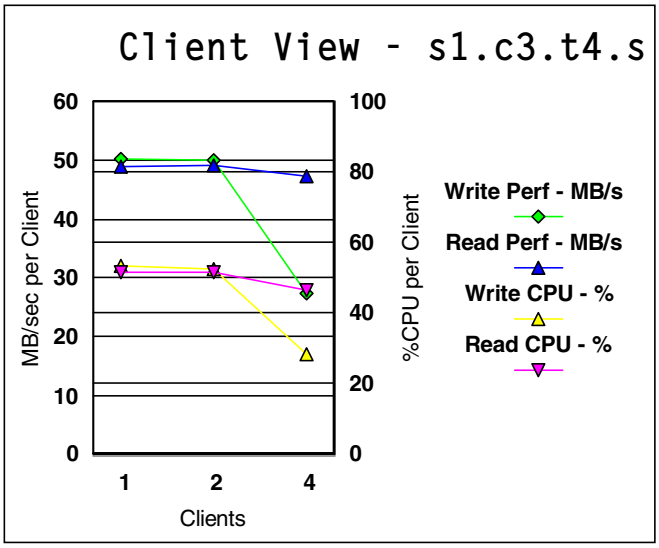


Figure 98. Performance graph s1.c3.t4 — Client view

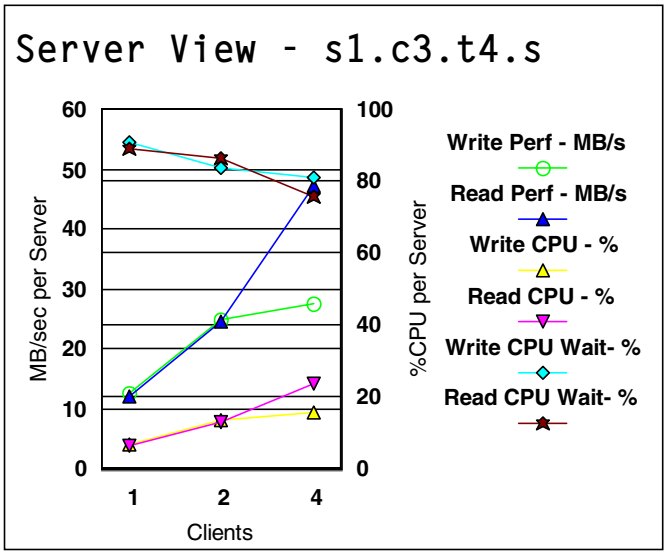


Figure 99. Performance graph s1.c3.t4 — Server view

A.1.1.15 S1_C3_Tests5

For the four clients run, the VSD servers are dropping IP packets causing VSD retries on most runs up to level 2, but the application does not fail.

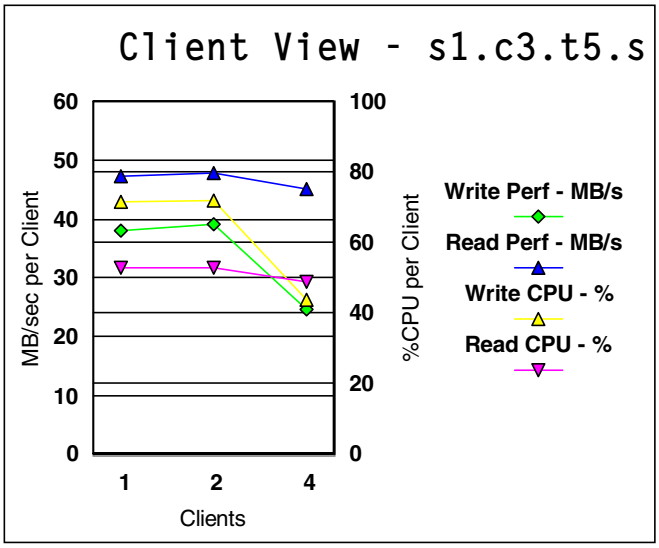


Figure 100. Performance graph s1.c3.t5 — Client view

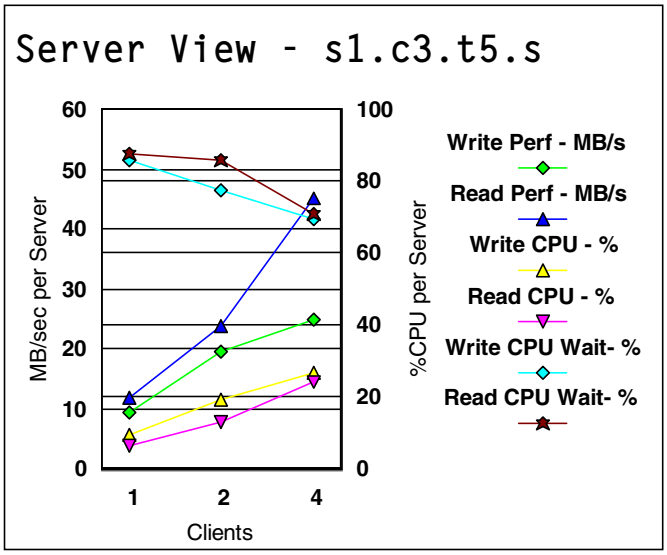


Figure 101. Performance graph s1.c3.t5 — Server view

A.1.1.16 S2_C1_Tests1

For this test, we did not get any azizo files for the client nodes. For the four clients run, both server v06n01i and v06n03 showed IP drops for most runs.

Figure 102. Performance graph s2.c1.t1 — Client view

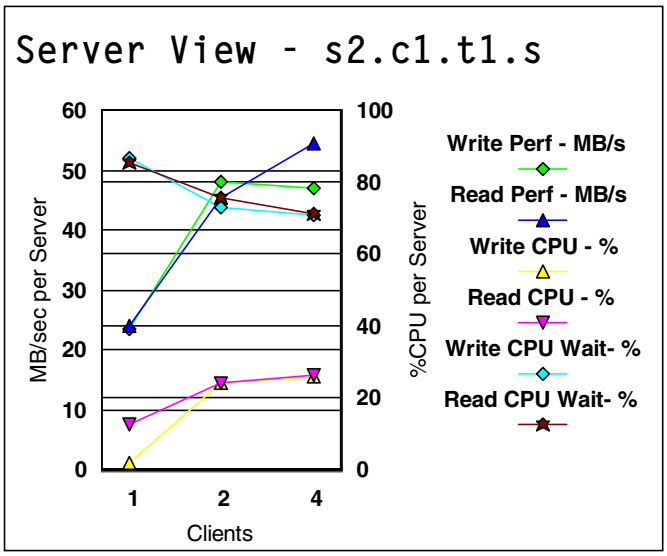


Figure 103. Performance graph s2.c1.t1 — Server view

A.1.1.17 S2_C1_Tests2

For this test, we did not get any azizo files for the client nodes. For the four clients run, server v06n01i showed IP drops for run 3.

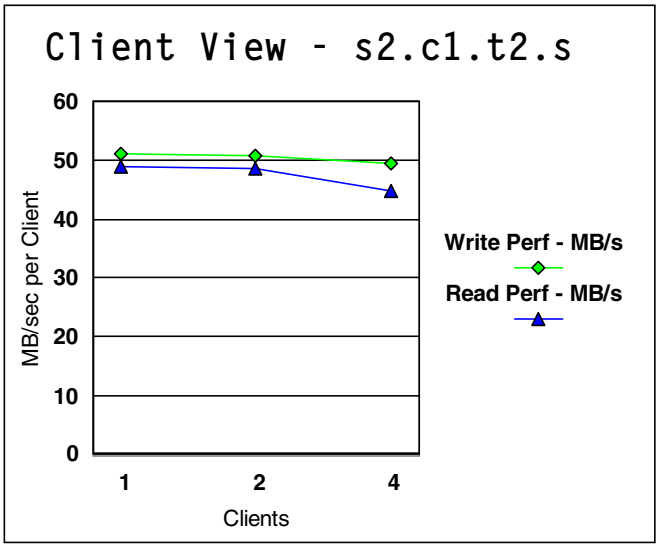


Figure 104. Performance graph s2.c1.t2 — Client view

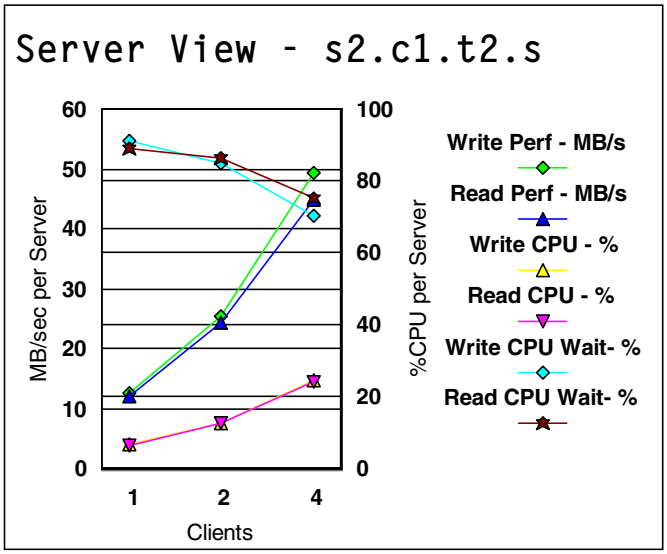


Figure 105. Performance graph s2.c1.t2 — Server view

A.1.1.18 S2_C1_Tests3

For this test, we did not get any azizo files for the client nodes.

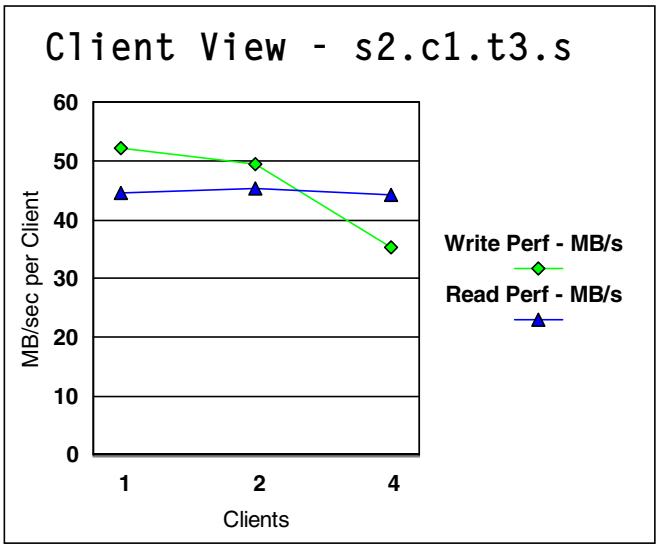


Figure 106. Performance graph s2.c1.t3 — Client view

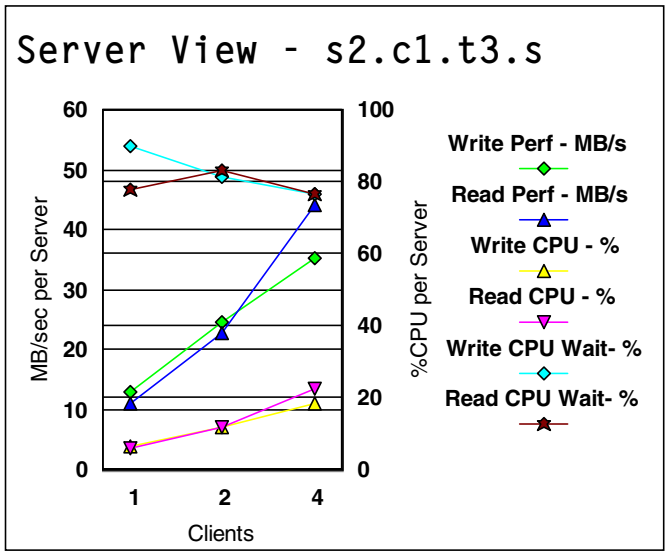


Figure 107. Performance graph s2.c1.t3 — Server view

A.1.1.19 S2_C1_Tests4

For this test, we did not get any azizo files for the client nodes. For the four clients run, server v06n01i showed IP drops for run 3.

Figure 108. Performance graph s2.c1.t4 — Client view

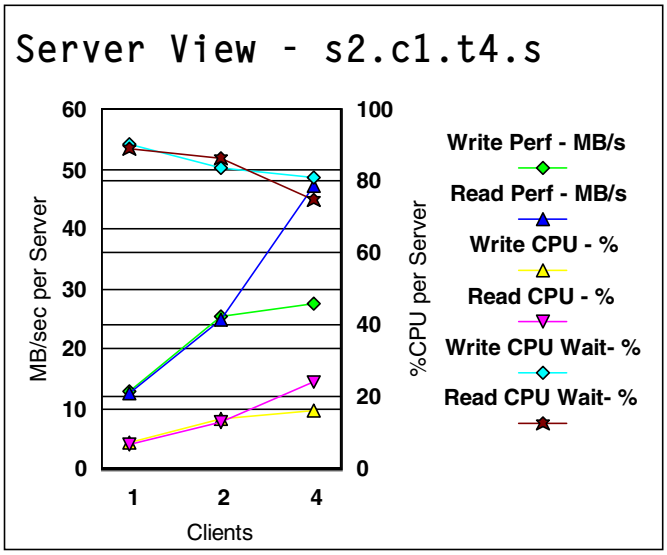


Figure 109. Performance graph s2.c1.t4 — Server view

A.1.1.20 S2_C1_Tests5

For this test, we did not get any azizo files for the client nodes. For the four clients run, all four servers showed IP drops for most runs.

Figure 110. Performance graph s2.c1.t5 — Client view

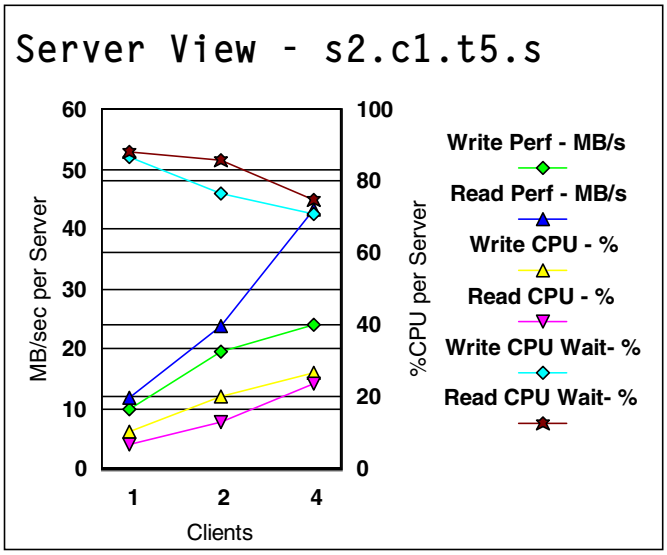


Figure 111. Performance graph s2.c1.t5 — Server view

A.1.2 Parallel

Presented below is the raw data from the Parallel Base run tests using the ior application.

A.1.2.1 S1_C4_Tests1

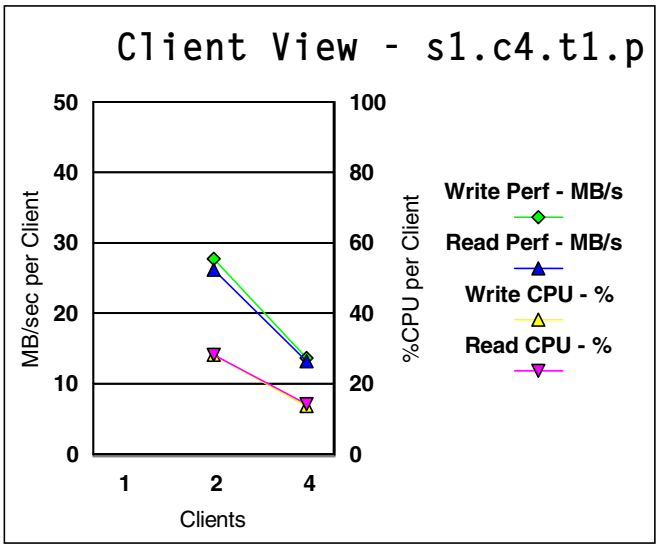


Figure 112. Performance graph s1.c4.t1 — Client view

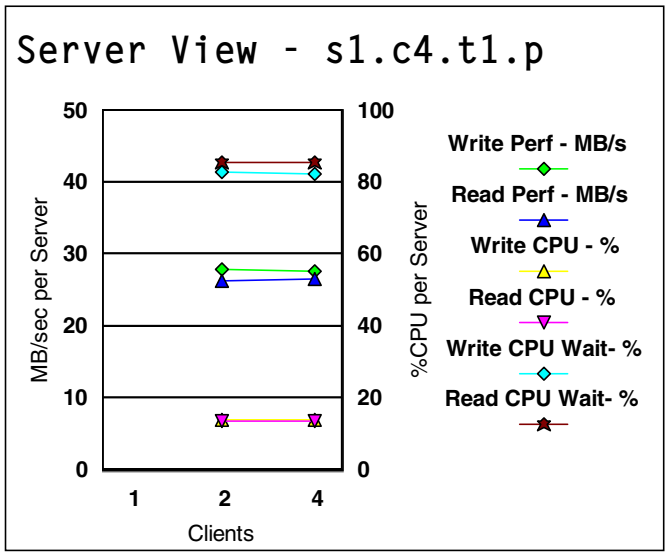


Figure 113. Performance graph s1.c4.t1 — Server view

A.1.2.2 S1_C4_Tests2

For the four client case, the application failed during run 1 due to VSD retries caused by IP packet drops mainly from VSD server v06n03.

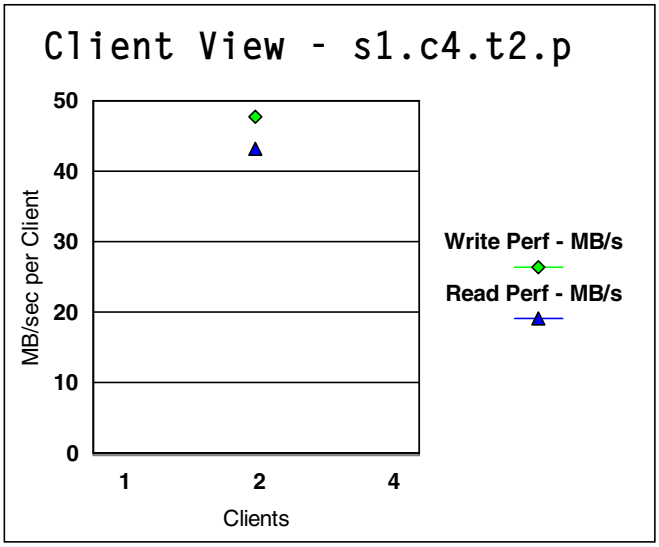


Figure 114. Performance graph s1.c4.t2 — Client view

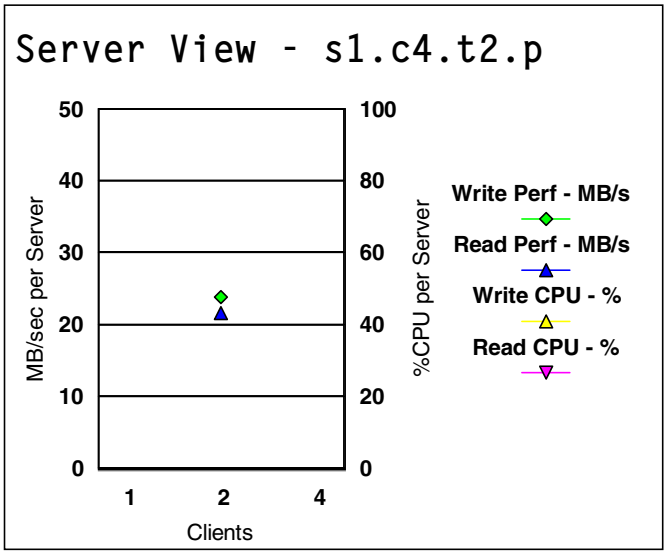


Figure 115. Performance graph s1.c4.t2 — Server view

A.1.2.3 S1_C4_Tests3

For the two client case, no azizo files were captured; so, no CPU information is available.

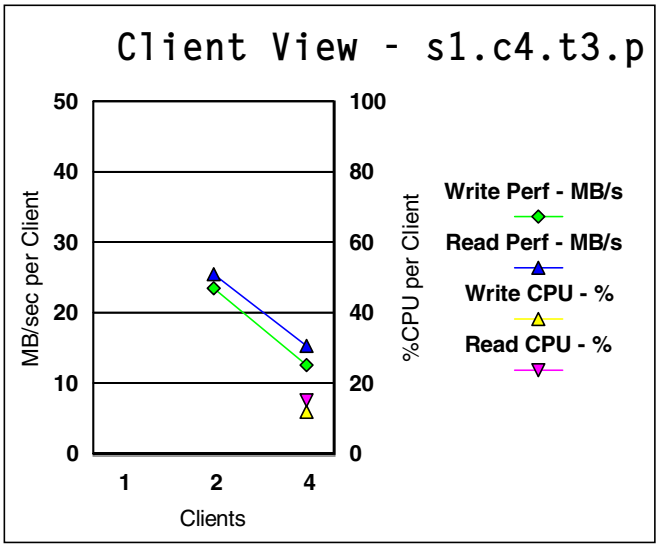


Figure 116. Performance graph s1.c4.t3 — Client view

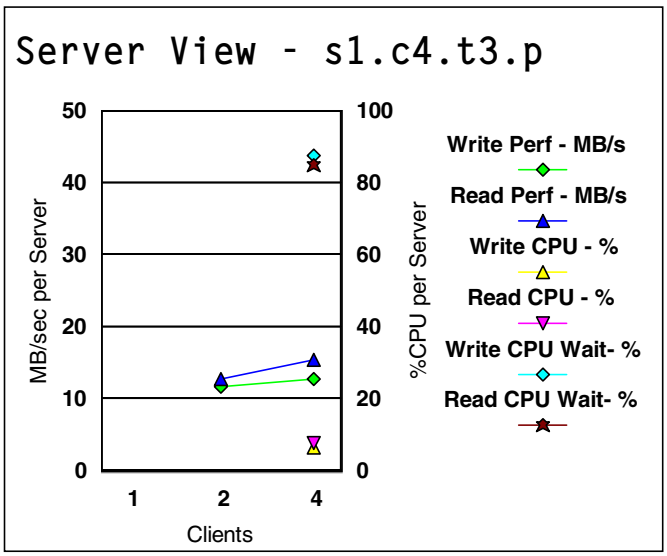


Figure 117. Performance graph s1.c4.t3 — Server view

A.1.2.4 S1_C4_Tests4

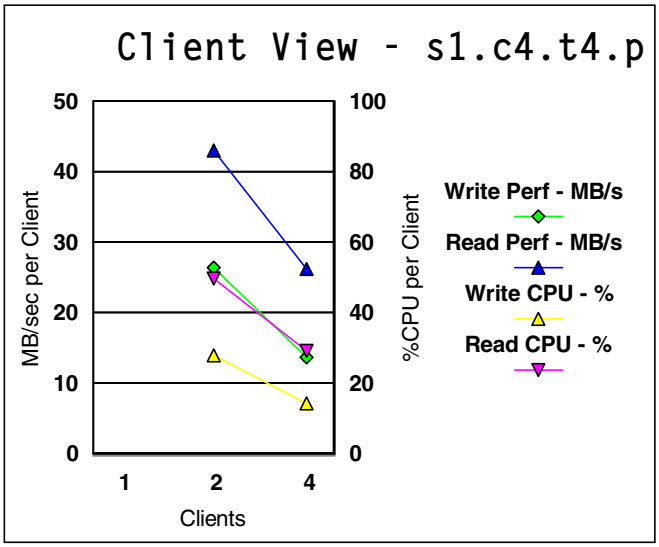


Figure 118. Performance graph s1.c4.t4 — Client view

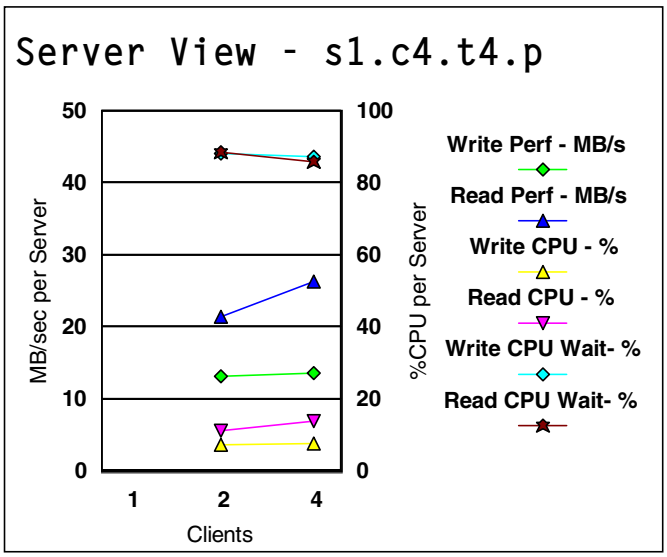


Figure 119. Performance graph s1.c4.t4 — Server view

A.1.2.5 S1_C4_Tests5

For the four client case, the application failed during run 3 due to VSD retries caused by IP packet drops mainly from VSD servers v06n05 and v06n07.

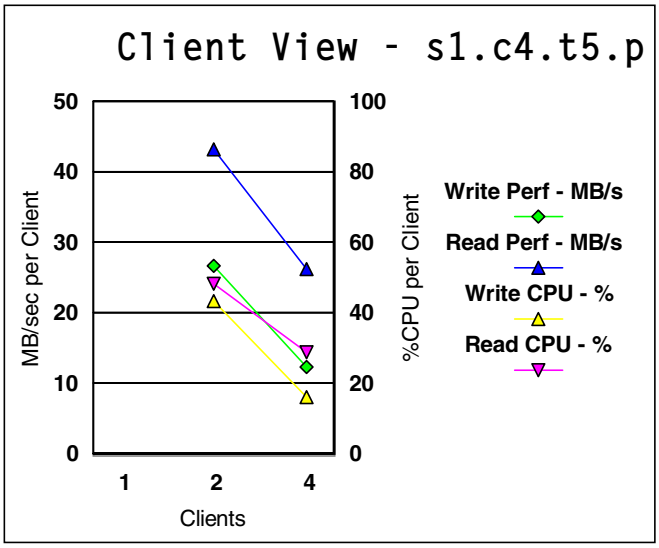


Figure 120. Performance graph s1.c4.t5 — Client view

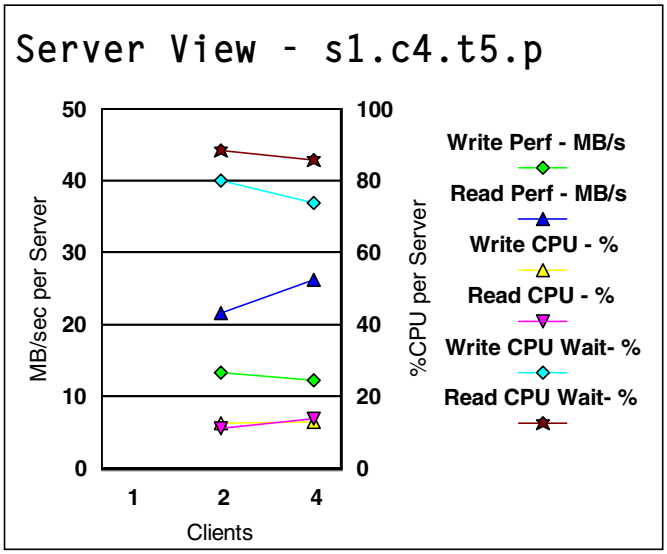


Figure 121. Performance graph s1.c4.t5 — Server view

A.1.2.6 S1_C2_Tests1

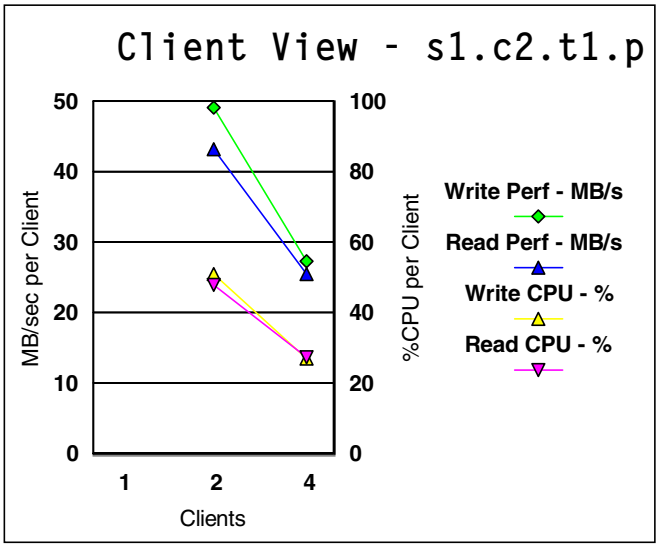


Figure 122. Performance graph s1.c2.t1 — Client view

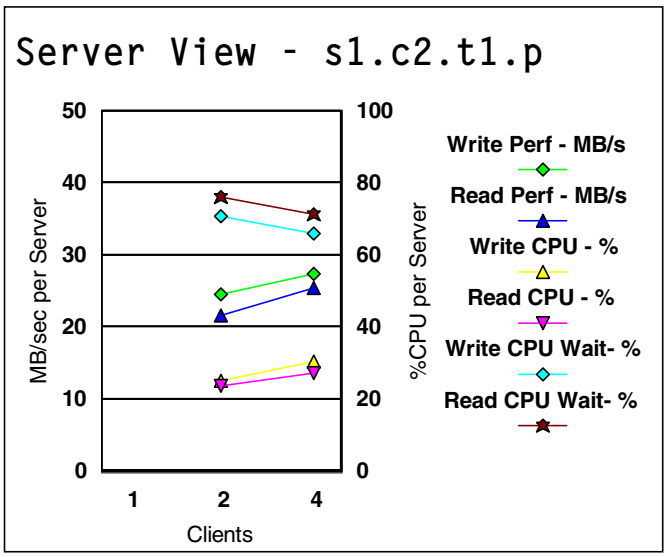


Figure 123. Performance graph s1.c2.t1 — Server view

A.1.2.7 S1_C2_Tests2

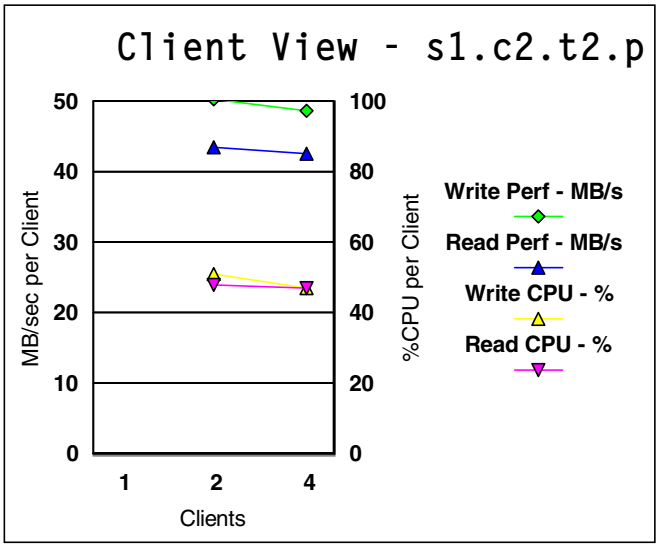


Figure 124. Performance graph s1.c2.t2 — Client view

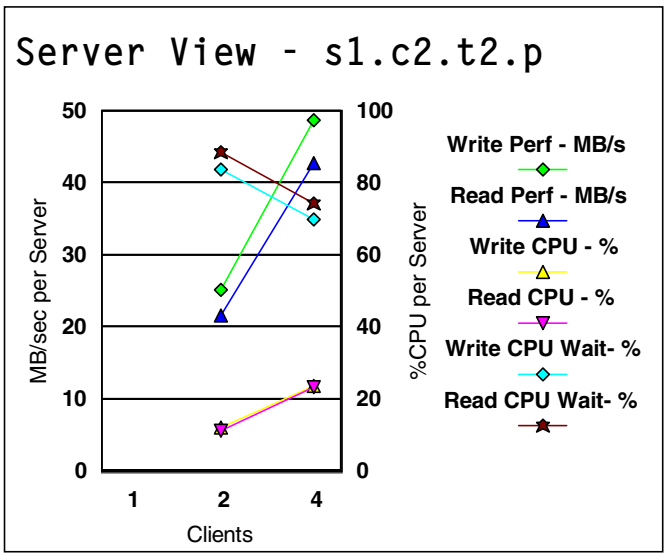


Figure 125. Performance graph s1.c2.t2 — Server view

A.1.2.8 S1_C2_Tests3

For the two clients run, one VSD retry was recorded for both GPFS clients v06n09 and v06n11.

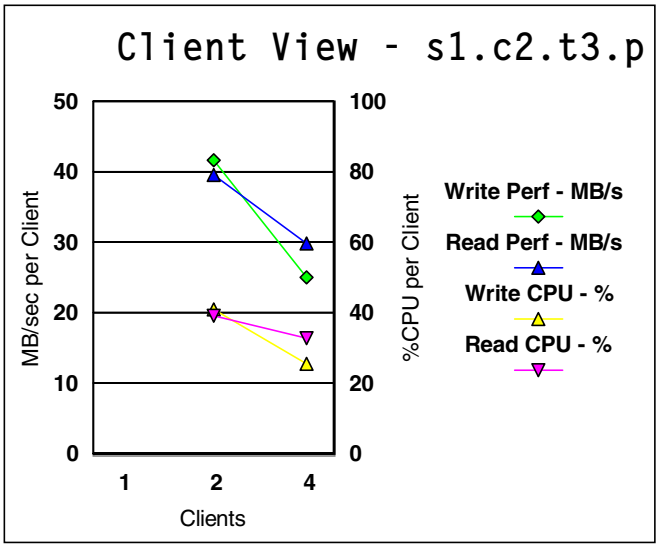


Figure 126. Performance graph s1.c2.t3 — Client view

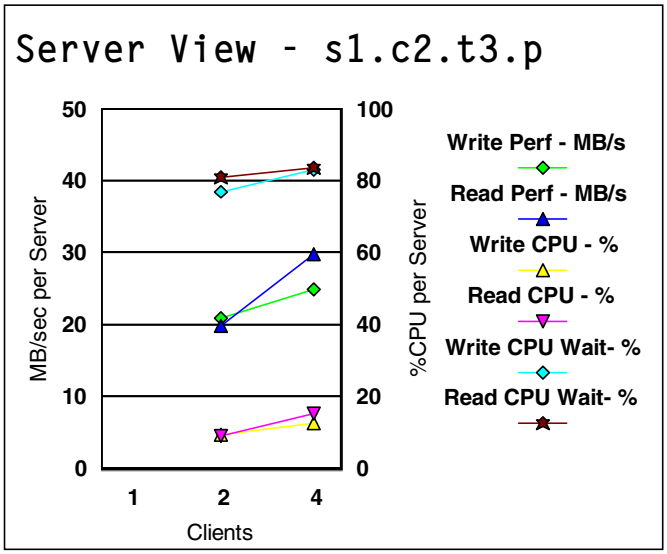


Figure 127. Performance graph s1.c2.t3 — Server view

A.1.2.9 S1_C2_Tests4

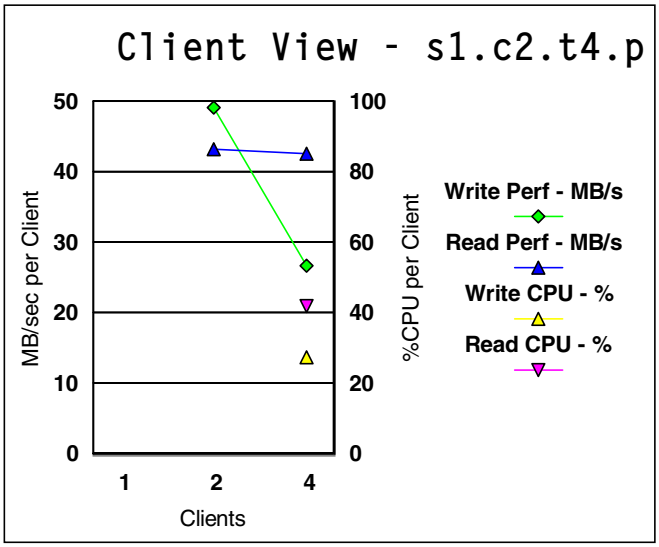


Figure 128. Performance graph s1.c2.t4 — Client view

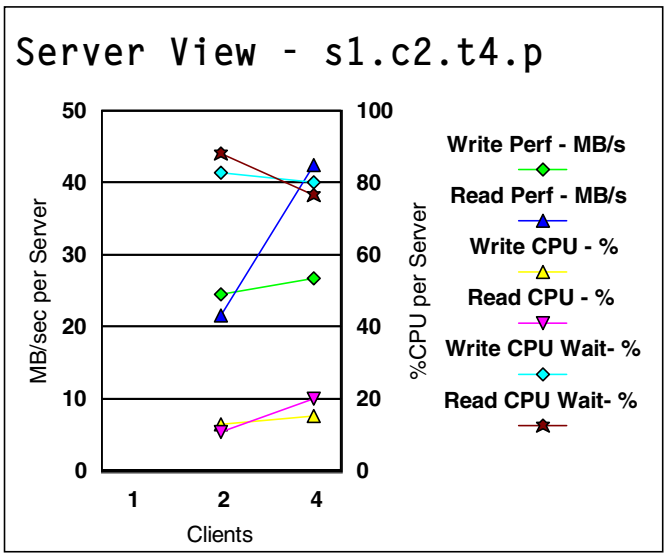


Figure 129. Performance graph s1.c2.t4 — Server view

A.1.2.10 S1_C2_Tests5

For the four client case, the VSD servers are dropping IP packets mainly on the writes. VSD retries were recorded up to level 2 for all the GPFS clients on most read and write runs. However, the application did not fail.

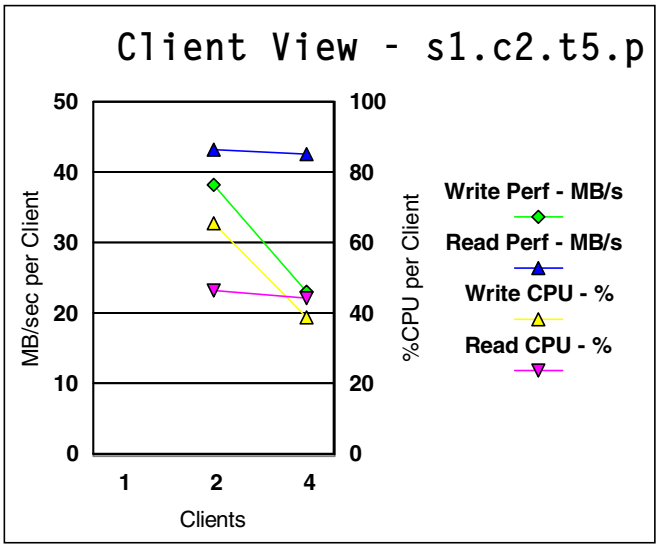


Figure 130. Performance graph s1.c2.t5 — Client view

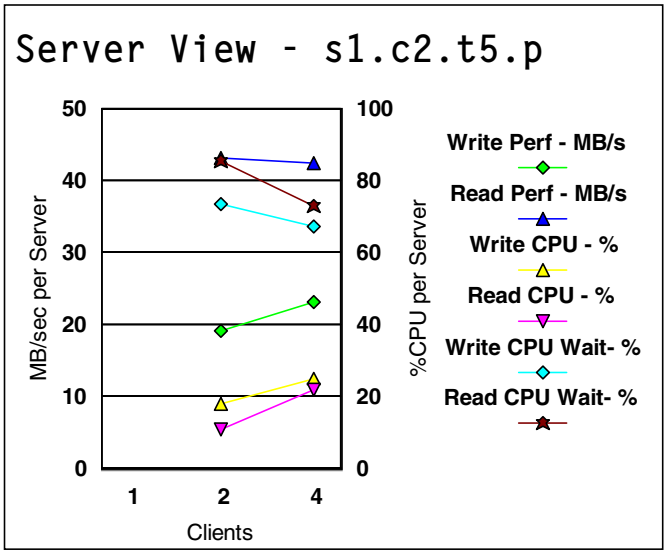


Figure 131. Performance graph s1.c2.t5 — Server view

A.1.2.11 S1_C3_Tests1

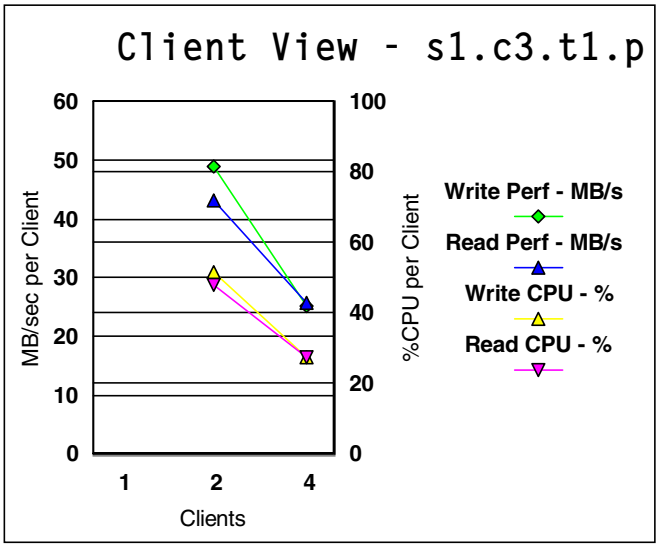


Figure 132. Performance graph s1.c3.t1 — Client view

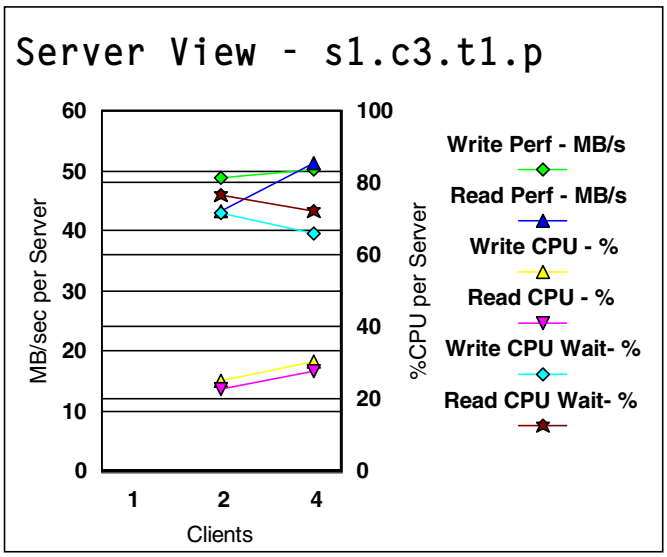


Figure 133. Performance graph s1.c3.t1 — Server view

A.1.2.12 S1_C3_Tests2

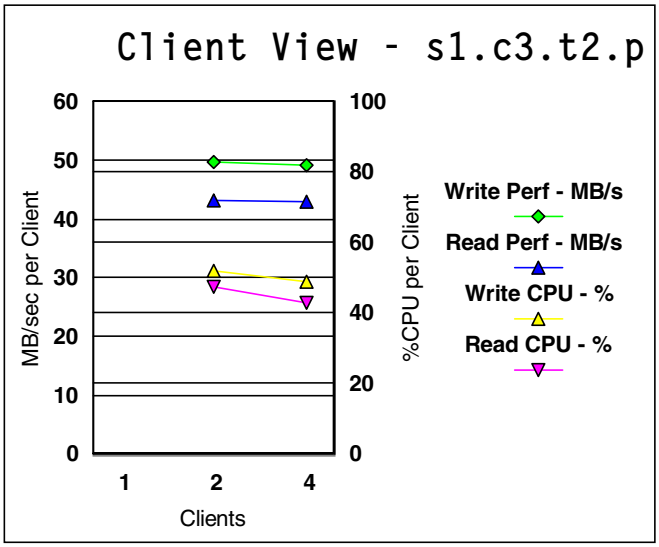


Figure 134. Performance graph s1.c3.t2 — Client view

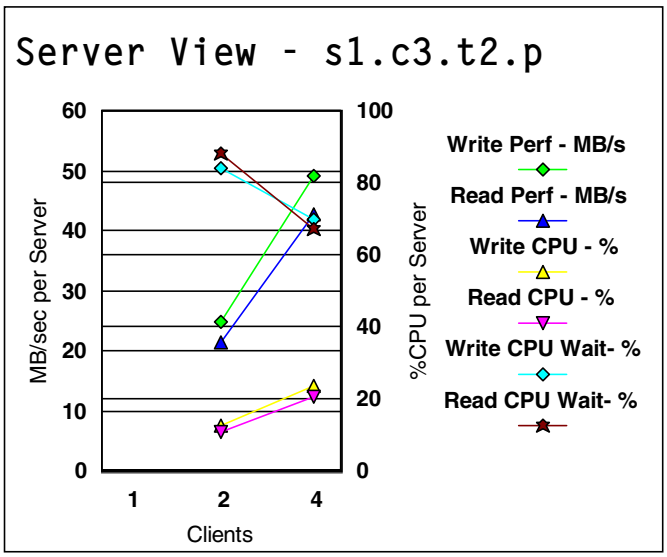


Figure 135. Performance graph s1.c3.t2 — Server view

A.1.2.13 S1_C3_Tests3

For the two clients run, one VSD retry level 1 was recorded on write run 5 for both GPFS clients v06n03 and v06n07. For the four clients run, one VSD retry level 1was recorded on read run 1 for GPFS client v06n03.

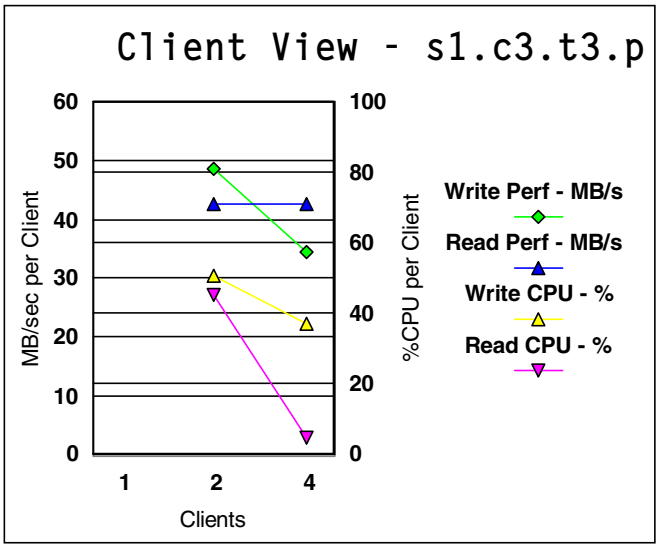


Figure 136. Performance graph s1.c3.t3 — Client view

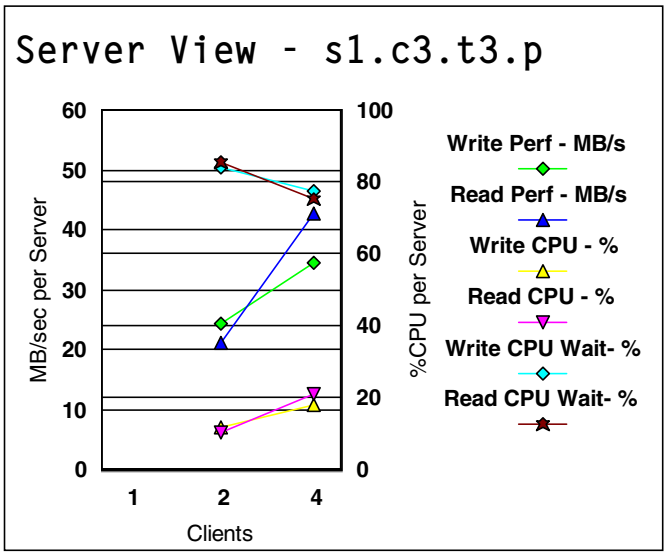


Figure 137. Performance graph s1.c3.t3 — Server view

A.1.2.14 S1_C3_Tests4

For the four client case, VSD retries up to level 1 were recorded on all GPFS client nodes for read run 2. IP packet drops were recorded for VSD server v06n05 for write run 2.

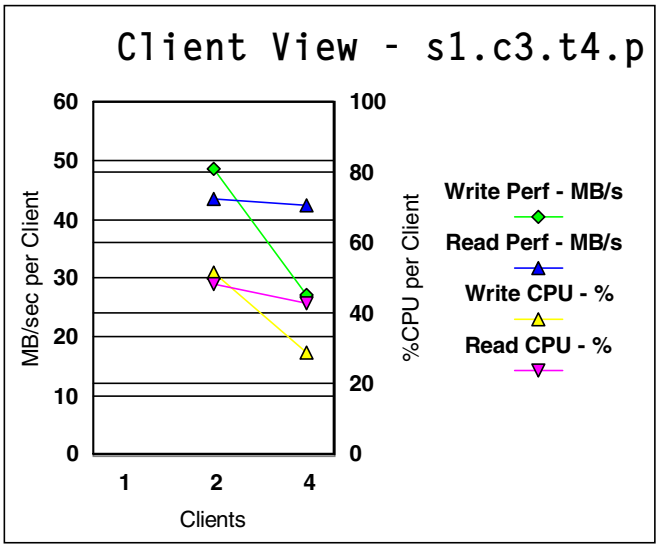


Figure 138. Performance graph s1.c3.t4 — Client view

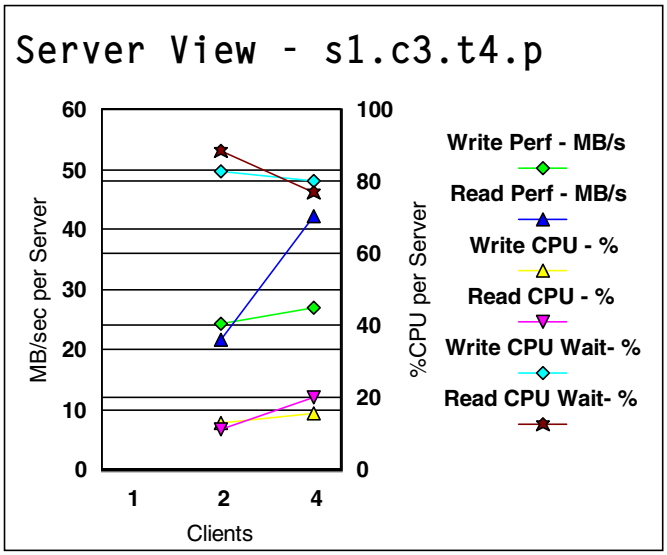


Figure 139. Performance graph s1.c3.t4 — Server view

A.1.2.15 S1_C3_Tests5

For the four client case, the application failed during run 1. VSD retries were recorded up to level 1 for all GPFS client nodes during write run 1.

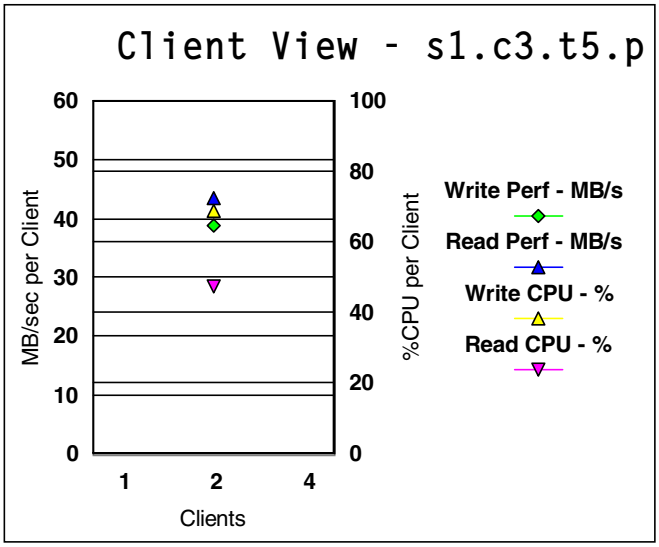


Figure 140. Performance graph s1.c3.t5 — Client view

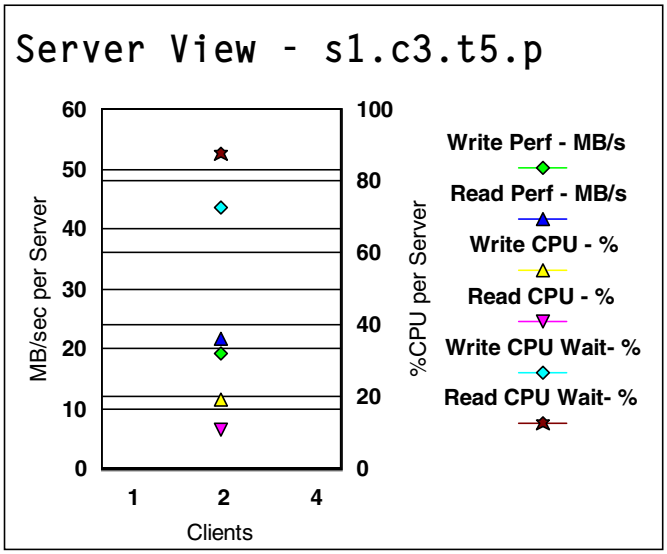


Figure 141. Performance graph s1.c3.t5 — Server view

A.1.2.16 S2_C1_Tests1

For this test, we did not get azizo files for client nodes. For the two clients run, IP drops were experienced by v06n01i on write run 1. For the four clients run, IP drops occurred during writes. Only one read and write run was completed.

Figure 142. Performance graph s2.c1.t1 — Client view

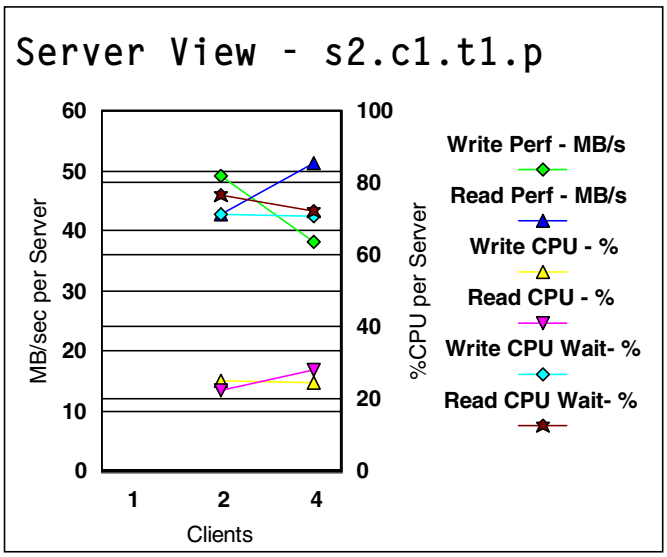


Figure 143. Performance graph s2.c1.t1 — Server view

A.1.2.17 S2_C1_Tests2

For this test, we did not get any azizo files for the client nodes and, hence, no CPU information.

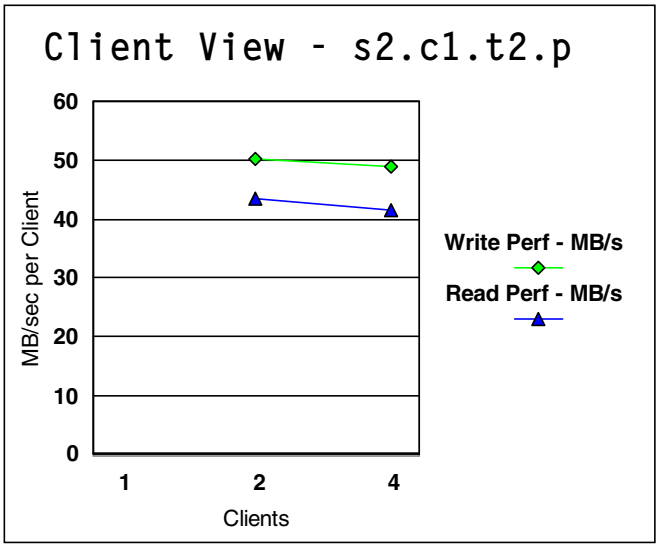


Figure 144. Performance graph s2.c1.t2 — Client view

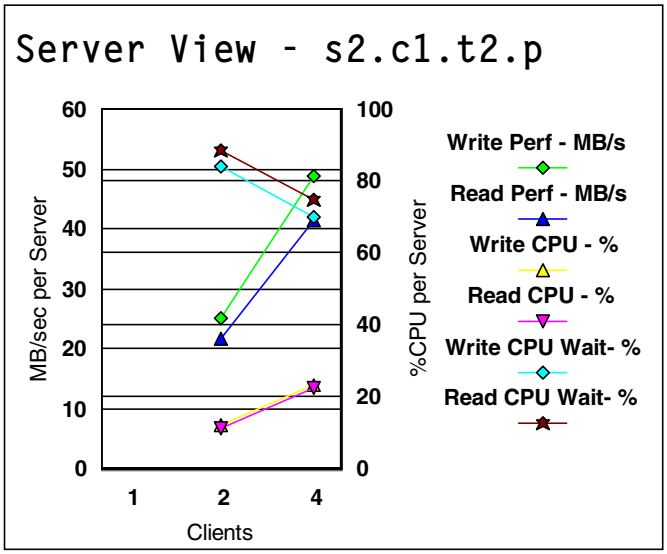


Figure 145. Performance graph s2.c1.t2 — Server view

A.1.2.18 S2_C1_Tests3

For this test, we did not get any azizo files for the client nodes and, hence, no CPU information. For the four clients run, IP drops were recorded during write run 1, but the application did not fail.

Figure 146. Performance graph s2.c1.t3 — Client view

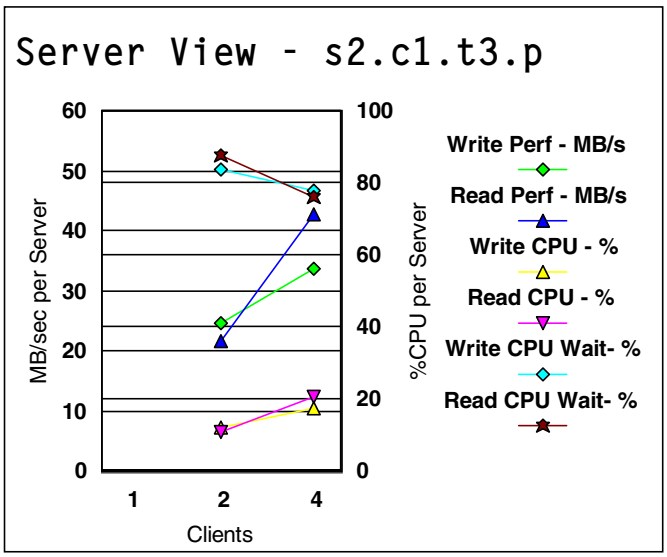


Figure 147. Performance graph s2.c1.t3 — Server view

A.1.2.19 S2_C1_Tests4

For this test we did not get any azizo files for the client nodes and, hence, no CPU information. For the four clients run, IP drops were recorded during write run 4, but the application did not fail.

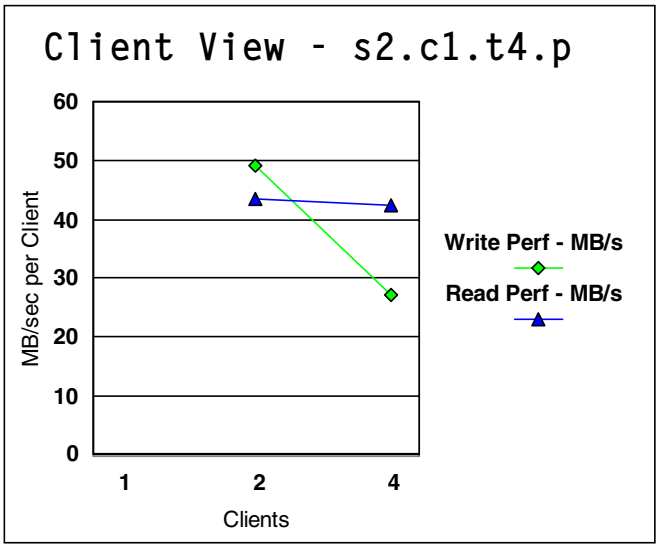


Figure 148. Performance graph s2.c1.t4 — Client view

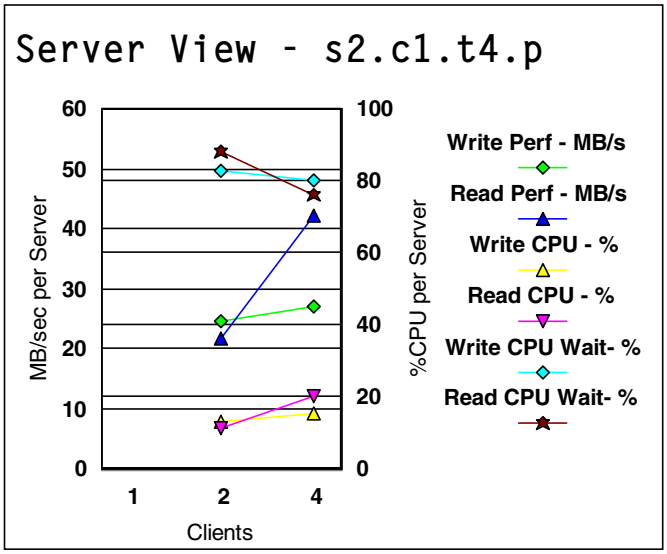


Figure 149. Performance graph s2.c1.t4 — Server view

A.1.2.20 S2_C1_Tests5

For this test we did not get any azizo files for the client nodes and, hence, no CPU information. For the four clients run, IP drops were recorded during all write runs and some read runs, yet the application did not fail.

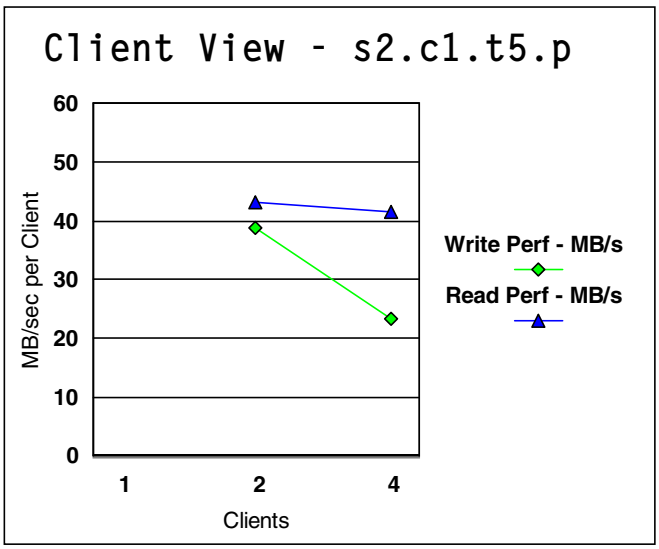


Figure 150. Performance graph s2.c1.t5 — Client view

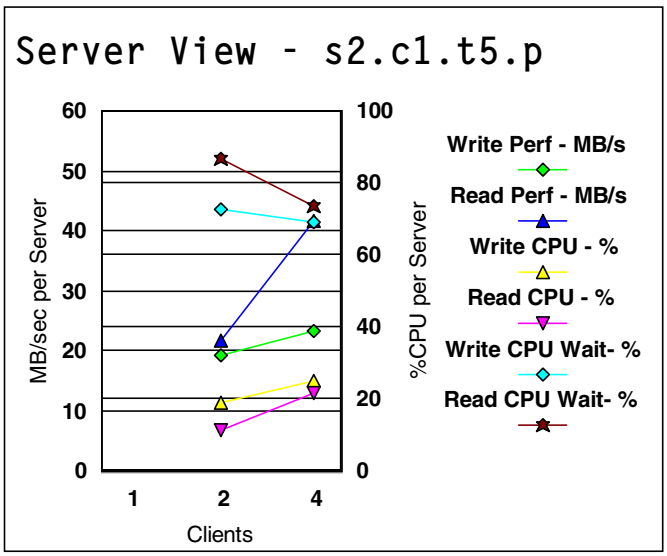


Figure 151. Performance graph s2.c1.t5 — Server view

A.1.3 Random

A.1.3.1 S1_C4_Tests1

The write CPU figures were much higher on v06n09. The stripe group manager was v06n13 throughout these runs.

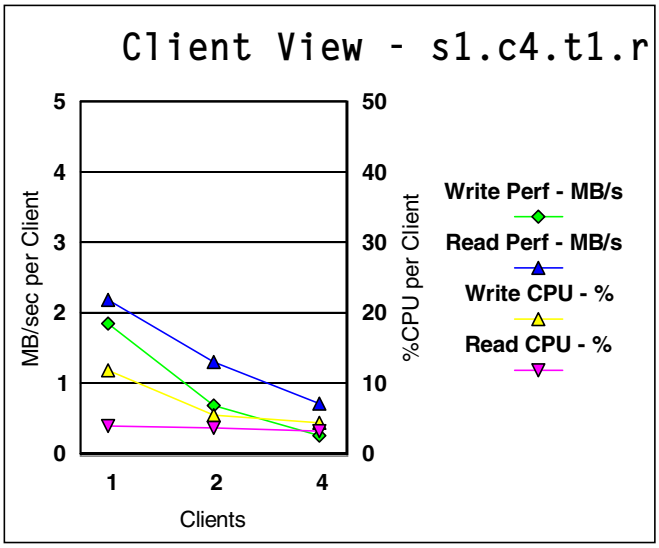


Figure 152. Performance graph s1.c4.t1 — Client view

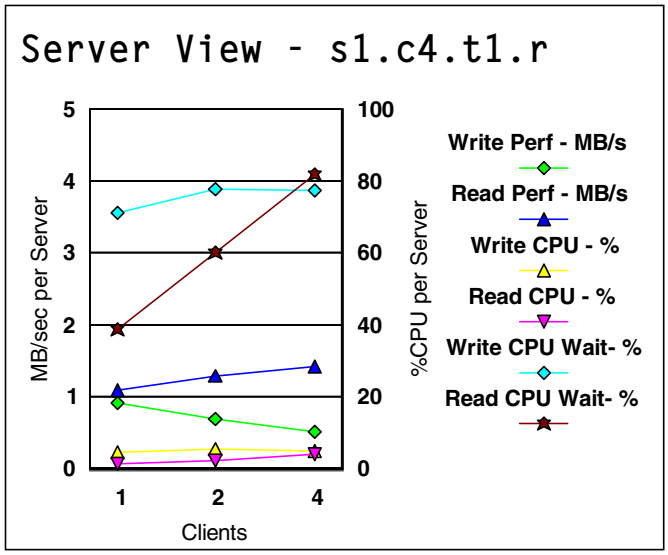


Figure 153. Performance graph s1.c4.t1 — Server view

A.1.3.2 S1_C4_Tests2

The client write CPU figures were much higher on v06n09 for some reason. The stripe group manager was v06n13 throughout these runs.

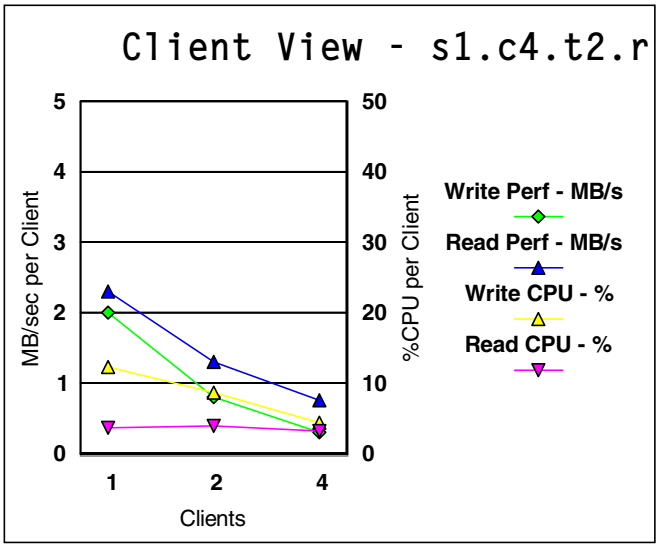


Figure 154. Performance graph s1.c4.t2 — Client view

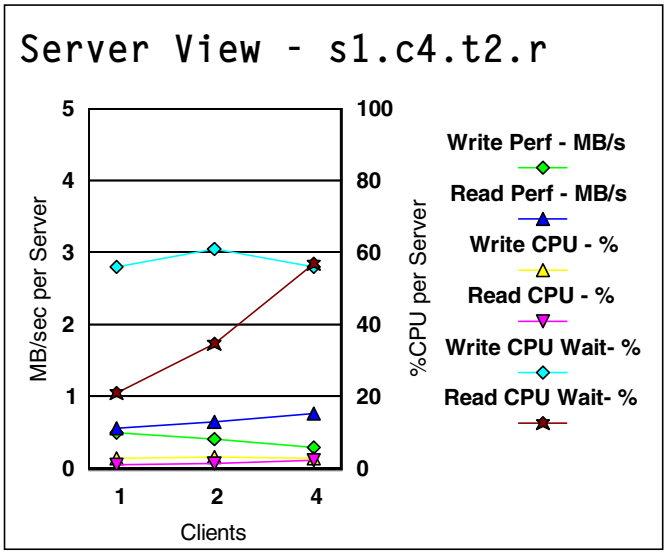


Figure 155. Performance graph s1.c4.t2 — Server view

A.1.3.3 S1_C4_Tests3

The client write CPU figures were much higher on v06n09 for some reason. The stripe group manager was v06n13 throughout these runs.

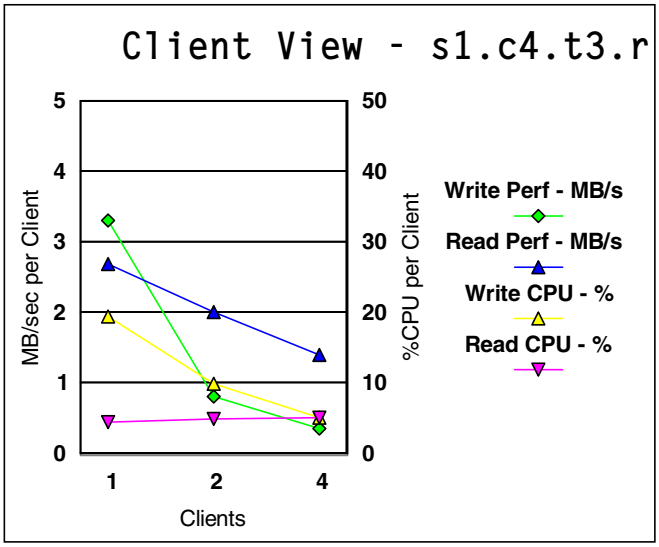


Figure 156. Performance graph s1.c4.t3 — Client view

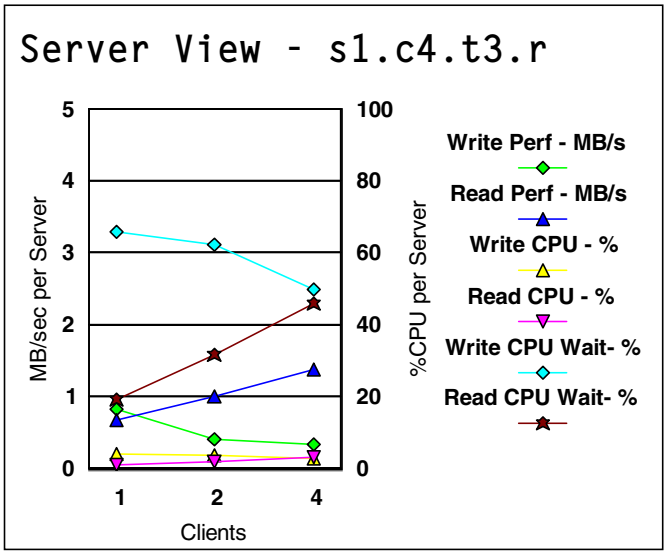


Figure 157. Performance graph s1.c4.t3 — Server view

A.1.3.4 S1_C4_Tests4

The client write CPU figures were much higher on v06n09 for some reason. The stripe group manager was v06n13 throughout these runs.

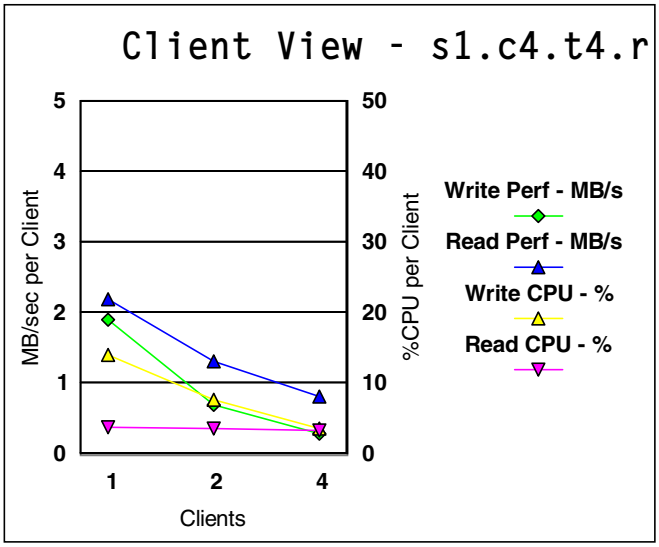


Figure 158. Performance graph s1.c4.t4 — Client view

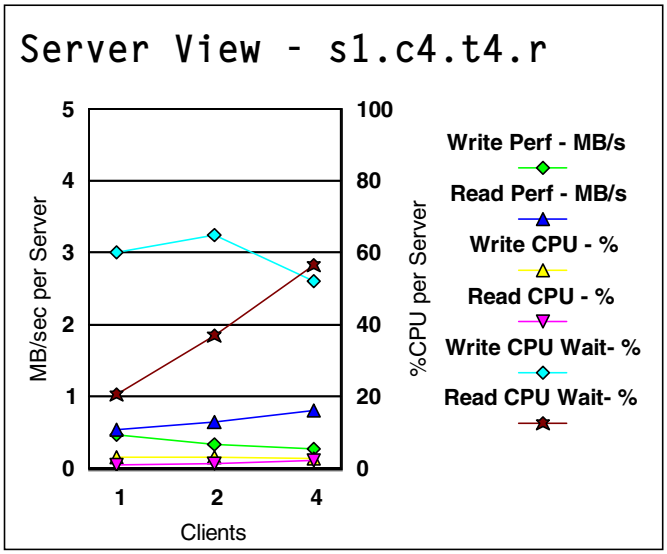


Figure 159. Performance graph s1.c4.t4 — Server view

A.1.3.5 S1_C4_Tests5

The client write CPU figures were much higher on v06n09 for some reason. The stripe group manager was v06n13 throughout these runs.

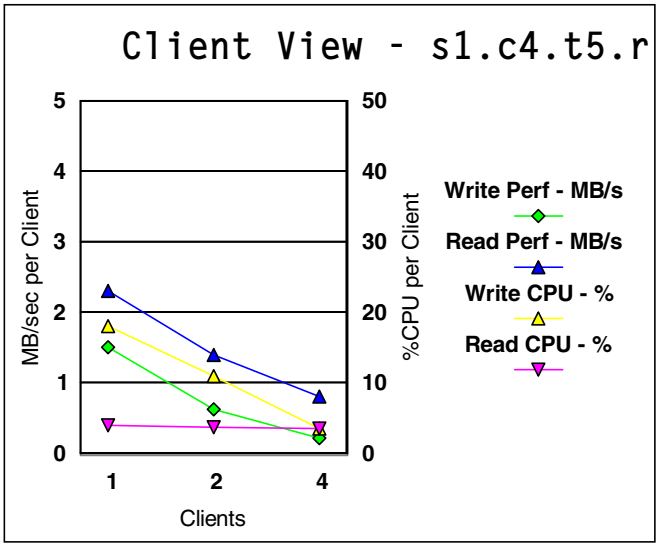


Figure 160. Performance graph s1.c4.t5 — Client view

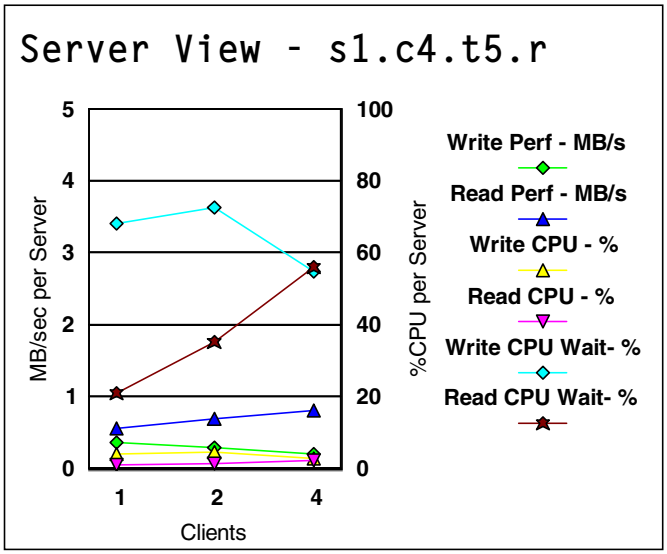
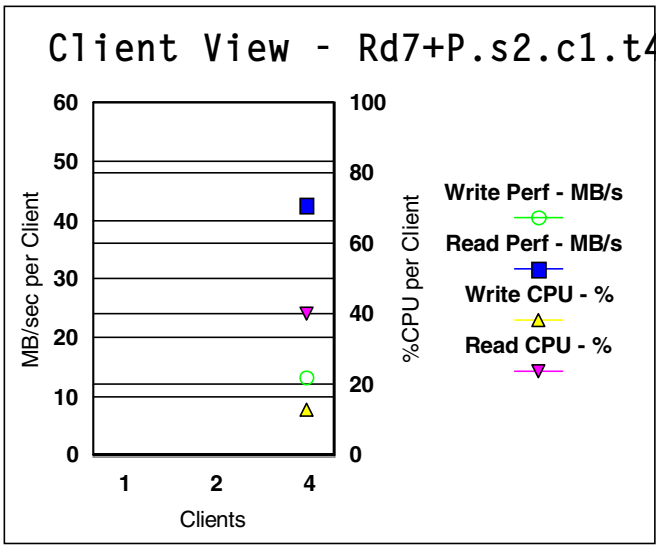


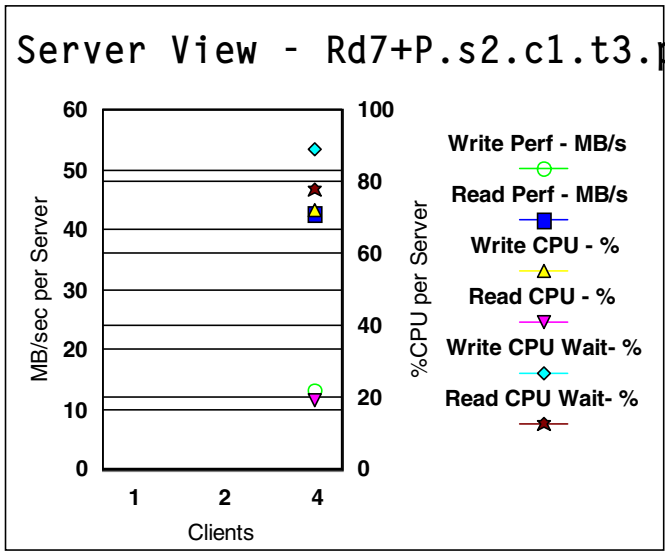
Figure 161. Performance graph s1.c4.t5 — Server view

A.2 RAID-5 array size tests

A.2.1 7+P RAID-5 S2_C1_Tests3

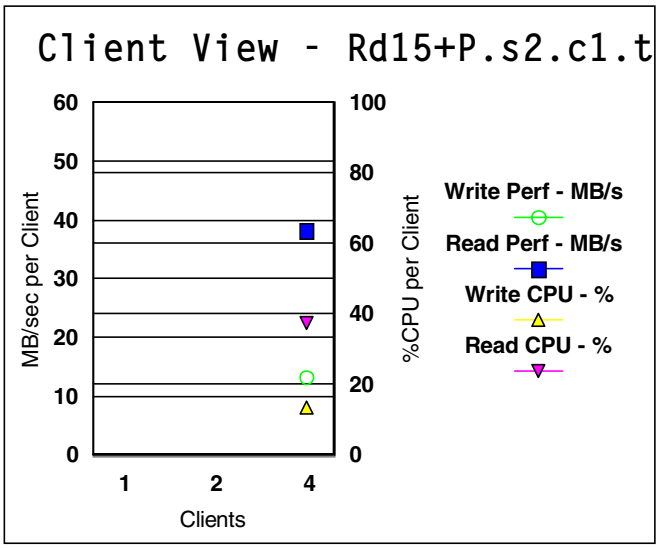


RAID-5 7+P performance graph s2.c1.t2 — Client view



RAID-5 7+P performance graph s2.c1.t2 — Server view

A.2.2 15+P RAID-5 S2_C1_Tests3



RAID-5 15+P performance graph s2.c1.t2 — Client view

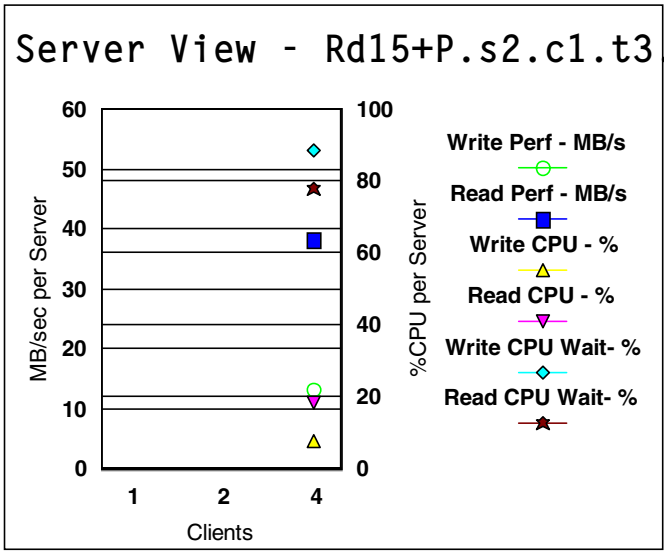


Figure 162. RAID-5 15+P performance graph s2.c1.t2 — Server view

A.3 Metadata tests

A.3.1 4+P RAID-5 combined data and metadata S2_C1_Tests3

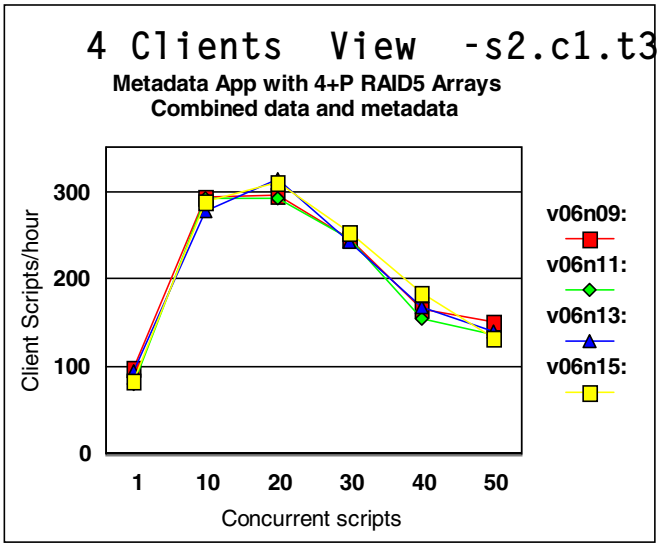


Figure 163. Combined metadata and data RAID-5 4+P s2.c1.t3 — 4 Clients view

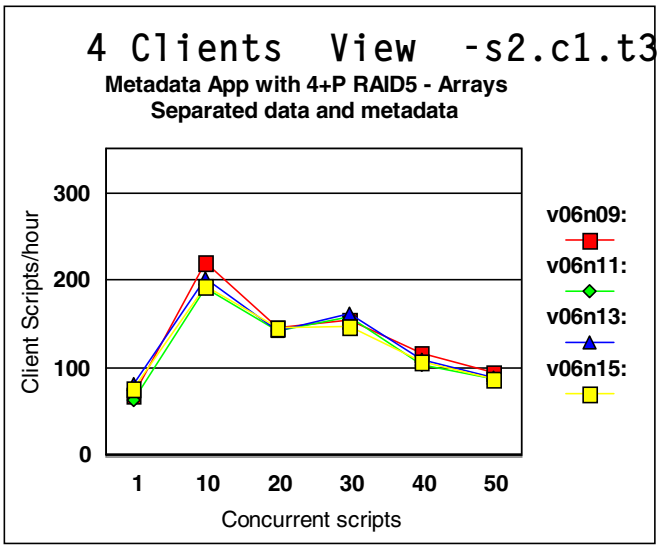


Figure 164. Separated metadata and data RAID-5 4+P s2.c1.t3 — 4 Clients view

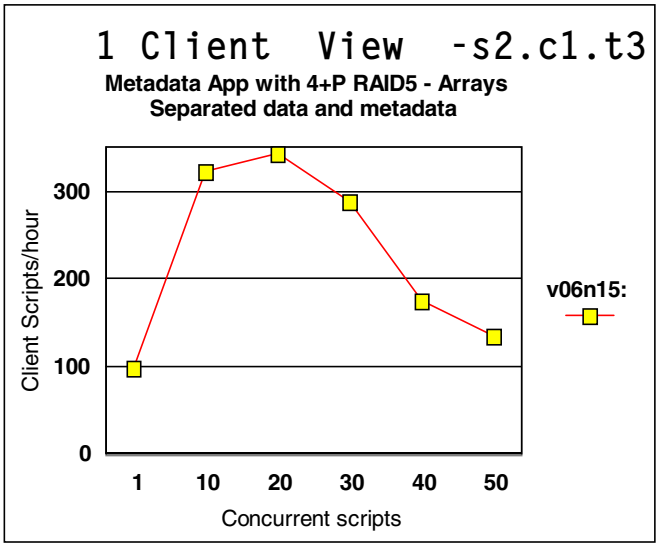


Figure 165. Separated metadata and data RAID-5 4+P s2.c1.t3 — 1 Client view

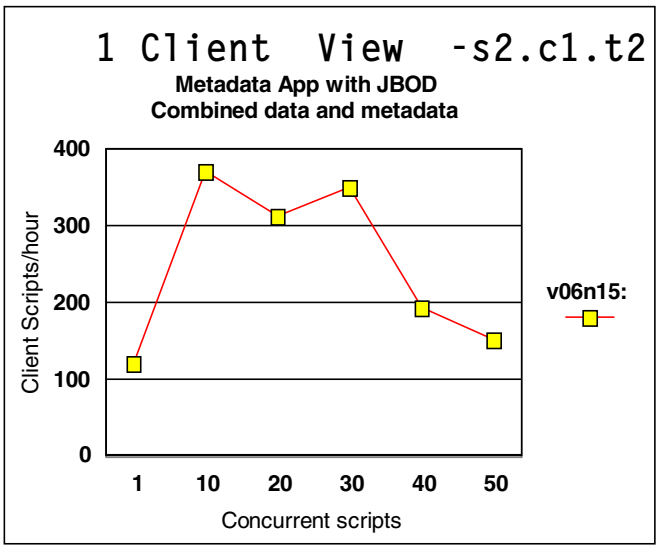


Figure 166. Combined metadata and data JBOD s2.c1.t2 — 1 Client view

A.4 Client max throughput tests

Presented below is the raw data from the Client max throughput tests.

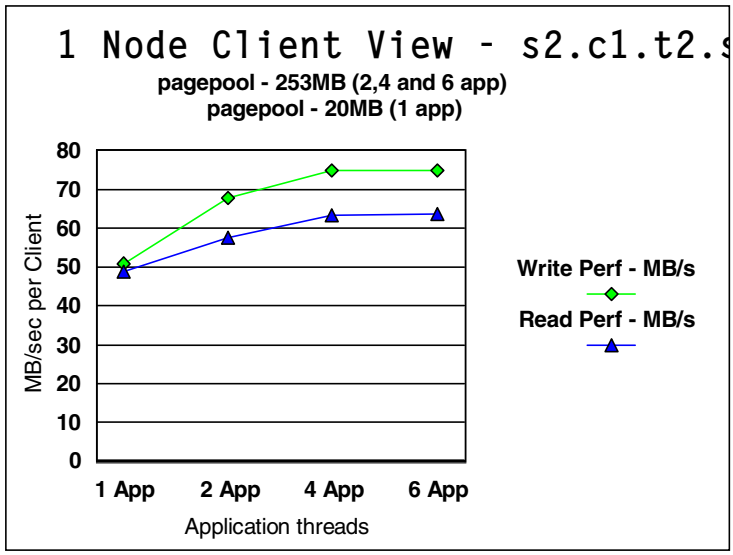


Figure 167. Single node multi application s2.c1.t2 —1 Node client view

Appendix B. Special notices

This publication is intended to help IBM Customers, Business Partners, IBM System Engineers, and other RS/6000 SP specialists who are involved in General Parallel File System, Version 1, Release 2 projects, including the education of RS/6000 SP professionals responsible for installing, configuring, and administering GPFS Version 1, Release 2. The information in this publication is not intended as the specification of any programming interfaces that are provided by General Parallel File System. See the PUBLICATIONS section of the IBM Programming Announcement for GPFS Version 1, Release 2 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, Northe 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 information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. 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.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

IBM ®	AIX
BookManager	Global Network
ESCON	HACMP/6000
LoadLeveler	OS/390
POWERparallel	RS/6000
S/390	SP

The following terms are trademarks of other companies:

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.

MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

SET and the SET logo are trademarks owned by SET Electronic Transaction LLC.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Appendix C. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

C.1 International Technical Support Organization publications

For information on ordering these ITSO publications see “How to get ITSO redbooks” on page 283.

- *GPFS: A Parallel File System*, SG24-5165
- *Understanding and Using the SP Switch*, SG24-5161
- *PSSP 3.1 Announcement*, SG24-5332
- *RS/6000 SP System Performance Tuning*, SG24-5340

C.2 Redbooks on CD-ROMs

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at `http://www.redbooks.ibm.com/` for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

C.3 Other publications

These publications are also relevant as further information sources:

- *General Parallel File System for AIX: Installation and Administration Guide*, SA22-7278

- *IBM Parallel System Support Programs for AIX: Managing Shared Disks*, SA22-7349
- *IBM Parallel System Support Programs for AIX: Performance Monitoring Guide and Reference*, SA22-7353
- *AIX Performance Toolbox User's Guide Version 1.2 and Version 2 for AIX: Guide and Reference*, SC23-2625

The following Web sites are also relevant as further information sources:

- `http://www.redbooks.ibm.com`
- `http://cs2.austin.ibm.com/ibmsm/ibmsm.nsf/mainframeset?readform`
- `http://www.tivoli.com`
- `http://www.elink.ibmlink.ibm.com/pbl/pbl`
- `http://w3.itso.ibm.com`
- `http://inews.ibm.com`

How to get ITSO redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** http://www.redbooks.ibm.com/

  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 via e-mail including information from the redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.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 for customer may be found at http://www.redbooks.ibm.com/ and for IBM employees at http://w3.itso.ibm.com/.

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 also view redbook, residency, and workshop announcements at http://inews.ibm.com/.

IBM Redbook 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.

List of abbreviations

AFS	Andrew File System
AIX	Advanced Interactive Executive
AMG	Adapter Membership Group
ANS	Abstract Notation Syntax
API	Application Programming Interface
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
BIS	Boot/Install Server
BOS	Base Operating System
BSD	Berkeley Software Distribution
BUMP	Bring-Up Microprocessor
CDE	Common Desktop Environment
CMI	Centralized Management Interface
CP	Crown Prince
CPU	Central Processing Unit
CSS	Communication Subsystem
CWS	Control Workstation
DASD	Direct Access Storage Devices
DB	Database
DCE	Distributed Computing Environment
DFS	Distributed File System
DNS	Domain Name Service
EM	Event Management
EMAPI	Event Management Application Programming Interface
EMCDB	Event Management Configuration Database
EMD	Event Manager Daemon
EPROM	Erasable Programmable Read-Only Memory
ESCON	Enterprise Systems Connection
FDDI	Fiber Distributed Data Interface
FIFO	First-in First-out
FTP	File Transfer Protocol
FS	File System
GB	Gigabytes
GL	Group Leader
GPFS	General Parallel File System
GS	Group Services
GSAPI	Group Services Application Programming Interface
GVG	Global Volume Group
HACMP	High Availability Cluster Multiprocessing
HACMP/ES	High Availability Cluster Multiprocessing Enhanced Scalability
HACWS	High Availability Control Workstation
hb	Heart Beat
HiPS	High Performance Switch
hrd	Host Respond Daemon

HSD Hashed Shared Disk

IBM International Business Machines Corporation

IP Internet Protocol

ISB Intermediate Switch Board

ISC Intermediate Switch Chip

ITSO International Technical Support Organization

JFS Journaled File System

JBOD Just a Bunch of Disk

LAN Local Area Network

LCD Liquid Crystal Display

LED Light Emitter Diode

LP Logical Partition

LRU Last Recently Used

LSC Link Switch Chip

LV Logical Volume

LVM Logical Volume Manager

MB Megabytes

MIB Management Information Base

MPI Message Passing Interface

MPL Message Passing Library

MPP Massive Parallel Processors

NFS Network File System

NIM Network Installation Management

NSB Node Switch Board

NSC Node Switch Chip

OID Object ID

ODM Object Data Manager

PAIDE Performance Aide for AIX

PE Parallel Environment

PID pRocess ID

POE Parallel Operating Environment

PP Physical Partition

PSSP Parallel System Support Programs

PTC Prepare to Commit

PTPE Performance Toolbox Parallel Extensions

PTX Performance Toolbox for AIX

PV Physical Volume

RAM Random Access Memory

RCP Remote Copy Protocol

RM Resource Monitor

RMAPI Resource Monitor Application Programming Interface

RPQ Request For Product Quotation

RSCT RS/6000 Cluster Technology

RSI Remote Statistics Interface

RVSD Recoverable Virtual Shared Disk

SBS Structured Byte String

SCSI Small Computer System Interface

SDR System Data Repository

SGM Stripe Group Manager

SMIT System Management Interface Tool

SSA	Serial Storage Architecture
VG	Volume Group
VSD	Virtual Shared Disk

Glossary

A

Adapter. An adapter is a mechanism for attaching parts. For example, an adapter could be a part that electrically or physically connects a device to a computer or to another device. In the SP system, network connectivity is supplied by various adapters, some optional, that can provide connection to I/O devices, networks of workstations, and mainframe networks. Ethernet, FDDI, token ring, HiPPI, SCSI, SSA, FCS, and ATM are examples of adapters that can be used as part of an SP system.

Address. A character or group of characters that identifies a register, a device, a particular part of storage, or some other data source or destination.

AFS. A distributed file system that provides authentication services as part of its file system creation.

AIX. Abbreviation for Advanced Interactive Executive, IBMs licensed version of the UNIX operating system. AIX is particularly suited to support technical computing applications including high function graphics and floating point computations.

API. Application Programming Interface. A set of programming functions and routines that provide access between the application layer of the OSI seven-layer model and applications that want to use the network. It is a software interface.

Application. The use to which a data processing system is put, for example, a payroll application, an airline reservation application, and so on.

Application Data. The data that is produced using an application program.

Authentication. The process of validating the identity of a user or server.

Authorization. The process of obtaining permission to perform specific actions.

B

Batch Processing. (1) The processing of data or the accomplishment of jobs accumulated in advance in such a manner that each accumulation, thus formed, is processed or accomplished in the same run. (2) The processing of data accumulating over a period of time. (3) Loosely, the execution of computer programs serially. (4) Computer programs executed in the background.

C

Client. (1) A function that requests services from a server and makes them available to the user. (2) A term used in an environment to identify a machine that uses the resources of the network.

CMI. Centralized Management Interface. provides a series of SMIT menus and dialogues used for defining and querying the SP system configuration.

Connectionless Network. A network in which the sending logical node must have the address of the receiving logical node before information interchange can begin. The packet is routed through nodes in the network based on the destination address in the packet. The originating source does not receive an acknowledgment that the packet was received at the destination.

Control Workstation. A single point of control allowing the administrator or operator to monitor and manage the SP system using the IBM AIX Parallel System Support Programs.

css. Communication subsystem.

D

Daemon. A process, not associated with a particular user, that performs system-wide functions, such as administration and control of networks, execution of time-dependent activities, line printer spooling, and so forth.

DASD. Direct Access Storage Device. Storage for input/output data.

DFS. Distributed File System. A subset of the IBM Distributed Computing Environment.

E

Ethernet. (1) Ethernet is the standard hardware for TCP/IP local area networks in the UNIX marketplace. It is a 10-megabit per second baseband type LAN that allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by collision detection (CSMA/CD). (2) A passive coaxial cable whose interconnections contain devices or components, or both, that are all active. It uses CSMA/CD technology to provide a best-effort delivery system.

F

Failover. The assuming of server responsibilities by the node designated as backup server when the primary server fails.

Failure Group. A collection of disks that share common access paths or adaptor connection and could all become unavailable through a single hardware failure.

Fall Back. Also called fallback, the sequence of events when a primary or server machine takes back control of its workload from a secondary or backup machine.

Fiber Distributed Data Interface (FDDI). An American National Standards Institute (ANSI) standard for 100-megabit-per-second LAN using optical fiber cables. An FDDI local area network (LAN) can be up to 100 km (62 miles) and can include up to 500 system units. There can be up to 2 km (1.24 miles) between system units and/or concentrators.

File Transfer Protocol (FTP). The Internet protocol (and program) used to transfer files between hosts. It is an application layer protocol in TCP/IP that uses TELNET and TCP protocols to transfer bulk-data files between machines or hosts.

File. A set of related records treated as a unit. For example, in stock control, a file could consist of a set of invoices.

File Name. A CMS file identifier in the form of 'filename filetype filemode (such as TEXT DATA A).

File Server. A centrally located computer that acts as a storehouse of data and applications for numerous users of a local area network.

Fragment. The space allocated an amount of data (usually the end of a file) too small to require a full block consisting of one or more subblocks (one thirty-second of block size).

G

Gateway. An intelligent electronic device interconnecting dissimilar networks and providing protocol conversion for network compatibility. A gateway provides transparent access to dissimilar networks for nodes on either network. It operates at the session presentation and application layers.

H

HACWS. High Availability Control Workstation function, based on HACMP, provides for a backup control workstation for the SP system.

Hashed Shared Disk (HSD). The data striping device for the IBM Virtual Shared Disk. The device driver lets application programs stripe data across physical disks in multiple IBM Virtual Shared Disks, thus, reducing I/O bottlenecks.

High Availability Cluster Multi-Processing. An IBM facility to cluster nodes or components to provide high availability by eliminating single points of failure.

Host. A computer connected to a network, providing an access method to that network. A host provides end-user services.

I

IBM Virtual Shared Disk. A subsystem that allows application programs executing on different nodes access to a raw logical volume as if it were local at each node.

i-node. The internal structure that describes an individual file to AIX. An i-node contains file size and update information as well as the addresses of data blocks, or in the case of large files,

indirect blocks that, in turn, point to data blocks. One i-node is required for each file.

Internet. A specific inter-network consisting of large national backbone networks, such as APARANET, MILNET, and NSFnet, and a myriad of regional and campus networks all over the world. The network uses the TCP/IP protocol suite.

Internet Protocol (IP). (1) A protocol that routes data through a network or interconnected networks. IP acts as an interface between the higher logical layers and the physical network. This protocol, however, does not provide error recovery, flow control, or guarantee the reliability of the physical network. IP is a connectionless protocol. (2) A protocol used to route data from its source to it destination in an Internet environment.

IP Address. A 32-bit address assigned to devices or hosts in an IP Internet that maps to a physical address. The IP address is composed of a network and host portion.

J

Journaled File System. The local file system within a single instance of AIX.

K

Kerberos. A service for authenticating users in a network environment.

Kernel. The core portion of the UNIX operating system which controls the resources of the CPU and allocates them to the users. The kernel is memory-resident, is said to run in *kernel mode*, and is protected by the hardware from user tampering.

L

LAN. (1) Acronym for Local Area Network, a data network located on the user's premises in which serial transmission is used for direct data communication among data stations. (2) Physical network technology that transfers data a high speed over short distances. (3) A network in which a set of devices is connected to another for communication and that can be connected to a larger network.

Local Host. The computer to which a user's terminal is directly connected.

Logical Volume Manager. Manages disk space at a logical level. It controls fixed-disk resources by mapping data between logical and physical storage allowing data to be discontiguous, span multiple disks, replicated, and dynamically expanded.

M

Metadata. Data structures that contain access information about file data. These might include i-nodes, indirect blocks, and directories. These data structures are used by GPFS but are not accessible to user applications.

Mirroring. The creation of a mirror image of data to be preserved in the event of disk failure.

N

Network. An interconnected group of nodes, lines, and terminals. A network provides the ability to transmit data to and receive data from other systems and users.

NFS. Network File System. NFS allows different systems (UNIX or non-UNIX), different architectures, or vendors connected to the same network to access remote files in a LAN environment as though they were local files.

O

ODM. Object Data Manager. In AIX, a hierarchical object-oriented database for configuration data.

P

Parallel Environment. A system environment where message passing or SP resource manager services are used by the application.

Parallel Environment. A licensed IBM program used for message passing applications on the SP or RS/6000 platforms.

Parallel Processing. A multiprocessor architecture that allows processes to be allocated to tightly coupled multiple processors in a cooperative processing environment allowing concurrent execution of tasks.

Parameter. (1) A variable that is given a constant value for a specified application and that may denote the application. (2) An item in a menu for which the operator specifies a value or for which the system provides a value when the menu is interpreted. (3) A name in a procedure that is used to refer to an argument that is passed to the procedure. (4) A particular piece of information that a system or application program needs to process a request.

Primary node or machine. (1) A device that runs a workload and has a standby device ready to assume the primary workload if that primary node fails or is taken out of service. (2) A node on the SP Switch that initializes, provides diagnosis and recovery services, and performs other operations to the switch network. (3) In IBM Virtual Shared Disk function, when physical disks are connected to two nodes (twin-tailed), one node is designated as the primary node for each disk, and the other is designated the secondary, or backup, node. The primary node is the server node for IBM Virtual Shared Disks defined on the physical disks under normal conditions. The secondary node can become the server node for the disks if the primary node is unavailable (off-line or down).

Primary Server. When physical disks are connected to two nodes (twin-tailed), this is the node that normally maintains and controls local access to the disk.

Process. (1) A unique, finite course of events defined by its purpose or by its effect, achieved under defined conditions. (2) Any operation or combination of operations on data. (3) A function being performed or waiting to be performed. (4) A program in operation. For example, a daemon is a system process that is always running on the system.

Protocol. A set of semantic and syntactic rules that defines the behavior of functional units in achieving communication.

Q

Quorum. The minimum number of nodes that must be running in order for the GPFS daemon to start. This is one plus half of the number of nodes in the GPFS configuration.

Quota. The amount of disk space and number of i-nodes assigned as upper limits for a specified user or group of users.

R

RAID. Redundant Array of Independent Disks. A set of physical disks that act as a single physical volume and use parity checking to protect against disk failure.

Recovery. The process of restoring access to file system data when a failure has occurred. This may involve reconstructing data or providing alternative routing through a different server.

Replication. The practice of creating and maintaining multiple file copies to ensure availability in the event of hardware failure.

RISC. Reduced Instruction Set Computing (RISC), the technology for today's high performance personal computers and workstations, was invented in 1975. Uses a small simplified set of frequently used instructions for rapid execution.

rlogin (remote LOGIN). A service offered by Berkeley UNIX systems that allows authorized users of one machine to connect to other UNIX systems across a network and interact as if their terminals were connected directly. The rlogin software passes information about the user's environment (for example, terminal type) to the remote machine.

RPC. Acronym for Remote Procedure Call, a facility that a client uses to have a server execute a procedure call. This facility is composed of a library of procedures plus an XDR.

RSH. A variant of `RLOGIN` command that invokes a command interpreter on a remote UNIX machine and passes the command line arguments to the command interpreter, thus, skipping the LOGIN step completely. See also rlogin.

S

SCSI. Small Computer Systems Interface. An adapter supporting attachment of various direct-access storage devices.

Secondary Node. In IBM Virtual Shared Disk function, when physical disks are connected to two nodes (twin-tailed), one node is designated as the primary node for each disk and the other is designated as the secondary, or backup, node. The secondary node acts as the server node for the IBM Virtual Shared disks defined on the physical disks if the primary node is unavailable (off-line or down).

Secondary Server. The second node connected to a twin-tailed disk. This node assumes control of local access if the primary server fails.

Server. (1) A function that provides services for users. A machine may run client and server processes at the same time. (2) A machine that provides resources to the network. It provides a network service, such as disk storage and file transfer, or a program that uses such a service. (3) A device, program, or code module on a network dedicated to providing a specific service to a network. (4) On a LAN, a data station that provides facilities to other data stations. Examples are file server, print server, and mail server.

Shell. The shell is the primary user interface for the UNIX operating system. It serves as command language interpreter, programming language, and allows foreground and background processing. There are three different implementations of the shell concept: Bourne, C, and Korn.

SMIT. The System Management Interface Toolkit is a set of menu driven utilities for AIX that provides functions, such as transaction login, shell script creation, automatic updates of object database, and so forth.

SNMP. Simple Network Management Protocol. (1) An IP network management protocol that is used to monitor attached networks and routers. (2) A TCP/IP-based protocol for exchanging network management information and outlining the structure for communications among network devices.

Socket. (1) An abstraction used by Berkeley UNIX that allows an application to access TCP/IP protocol functions. (2) An IP address and port number pairing. (3) In TCP/IP, the Internet address of the host computer on which the application runs and the port number it uses. A TCP/IP application is identified by its socket.

SSA. Serial Storage Architecture. An expanded storage adapter for multi-processor data sharing in UNIX-based computing allowing disk connection in a high-speed loop.

Standby Node or Machine. A device that waits for a failure of a primary node in order to assume the identity of the primary node. The standby machine then runs the primary's workload until the primary is back in service.

Stripe Group. A file system written across many disks, which are connected to multiple nodes.

Striping. A method of writing a file system, in parallel, to multiple disks instead of to single disks in a serial operation.

Sub-block. The smallest unit of data accessible in an I/O operation equal to one thirty-second of a data block.

Subnet. Shortened form of subnetwork.

Subnet Mask. A bit template that identifies to the TCP/IP protocol code the bits of the host address that are to be used for routing for specific subnetworks.

Subnetwork. Any group of nodes that have a set of common characteristics, such as the same network ID.

Subsystem. A software component that is not usually associated with a user command. It is usually a daemon process. A subsystem will perform work or provide services on behalf of a user request or operating system request.

Sysctl. Secure System Command Execution Tool. An authenticated client/server system for running commands remotely and in parallel.

System Partition. A group of non-overlapping nodes on a switch chip boundary that act as a logical SP system.

T

tar. Tape ARchive, is a standard UNIX data archive utility for storing data on tape media.

TCP. Acronym for Transmission Control Protocol, a stream communication protocol that includes error recovery and flow control.

TCP/IP. Acronym for Transmission Control Protocol/Internet Protocol, a suite of protocols designed to allow communication between networks regardless of the technologies implemented in each network. TCP provides a reliable host-to-host protocol between hosts in packet-switched communications networks and in interconnected systems of such networks. It assumes that the underlying protocol is the Internet Protocol.

Telnet. Terminal Emulation Protocol, a TCP/IP application protocol that allows interactive access to foreign hosts.

Token Management. A system for controlling file access in which each application performing a read or write operation is granted exclusive access to a specific block of file data. This ensures data consistency and controls conflicts.

Token Ring. (1) Network technology that controls media access by passing a token (special packet or frame) between media-attached machines. (2) A network with a ring topology that passes tokens from one attaching device (node) to another. (3) The IBM Token Ring LAN connection allows the RS/6000 system unit to participate in a LAN adhering to the IEEE 802.5 Token Passing Ring standard or the ECMA standard 89 for Token Ring, baseband LANs.

Transaction. An exchange between the user and the system. Each activity the system performs for the user is considered a transaction.

U

UNIX Operating System. An operating system developed by Bell Laboratories that features multiprogramming in a multiuser environment. The UNIX operating system was originally developed for use on minicomputers but has been adapted for mainframes and microcomputers. Note: The AIX operating system is IBMs implementation of the UNIX operating system.

User. Anyone who requires the services of a computing system.

User Datagram Protocol (UDP). (1) In TCP/IP, a packet-level protocol built directly on the Internet Protocol layer. UDP is used for application-to-application programs between TCP/IP host systems. (2) A transport protocol in the Internet suite of protocols that provides unreliable, connectionless datagram service. (3) The Internet Protocol that enables an application programmer on one machine or process to send a datagram to an application program on another machine or process.

User ID. A non-negative integer, contained in an object of type uid_t, that is used to uniquely identify a system user.

V

Virtual Shared Disk, IBM. The function that allows application programs executing at different nodes of a system partition to access a raw logical volume as if it were local at each of the nodes. In actuality, the logical volume is local at only one of the nodes (the server node).

W

Workstation. (1) A configuration of input/output equipment at which an operator works. (2) A terminal or microcomputer, usually one that is connected to a mainframe or to a network, at which a user can perform applications.

X

X Window System. A graphical user interface product.

Index

Numerics

15+P 199
4+P 199
6215 SSA Adapter 91
7+P 199

A

Access Control Lists 18
  ADSM support 166
ACLs
  See Access Control Lists 166
adapter loops 98
adapters 91
ADSM
  See ADSTAR Distributed Storage Manager 165
ADSTAR Distributed Storage Manager
  accessing GPFS metadata 168
  ADSM client on single VSD client node 175
  ADSM clients on VSD server nodes 187
  backing up to and restoring from tape 197
  backup-archive support of GPFS 165
  client and server on one SP node 180
  client options 174
  comparing full incremental with selective backup 192
  comparing JFS with GPFS 191
  comparing restore with replace 193
  full incremental backup 167
  good network performance due to SP Switch 196
  handling of large file systems and files on GPFS 168
  I/O rate of server 168
  impact of delayed update 167
  impact of tuning maxFilesToCache on performance 189
  incremental-by-date 167
  LARGECOMMBUFFERS option 174
  locating ADSM clients within SP environment 196
  locating ADSM server within SP environment 196
  point-in-time restore 168
  Reference Configuration 175
  resource requirements for implementation on GPFS 171
  restore 168
  restore IFNEWER 168
  restore REPLACE 173, 193
  running ADSM client on multiple SP nodes 183, 197
  running multiple ADSM client sessions 197
  selective backup 167
  serialization 167
  server options 174
  shared memory communication 180
  storage pool volumes of server setup 172
  support of GPFS ACLs 166
  TCPBUFSIZE option 174
  TCPNODELAY option 174
  TCPWINDOWSIZE option 174
  test data 173
  test methodology 173
  test system configuration for testing 172
  Tivoli Storage Management 165
  tuning maxFilesToCache for ADSM performence 191
  TXNBYTELIMIT option 174
  TXNGROUPMAX option 174
allocation segment 6
Andrew File System 50
Application
  Block Size 57
  I/O patterns 58
  Parallel I/O 59
  Portability 58
  Random I/O 58
  Sequential I/O 58
  Serial I/O 59
application block size 87, 88
application programming interface 1
applications I/O access method 199

B

Base Run tests 199
block size 87
byte-range locking 2

C

caching 2

case study
GPFS Configuration 151
GPFS file system configuration 154
Hardware Configuration 151
Parallel Test on a JBOD GPFS 155
Parallel Test on a Replicated GPFS 158
Random Test on a JBOD GPFS 157
Random Test on a Replicated GPFS 162
Software Configuration 151
tunable parameters 152
Variable Block Size on GPFS 164
client throughput 87
collective 64
Commands
cfgtb3 26
dd 146
Eclock 31
errpt 137
Estart 30
full incremental backup 167
incremental-by-date 167
iostat 143, 144
lsdev 27
lsvg 143
netstat 135, 138, 145
no 138
point-in-time restore 168
restore 168
restore IFNEWER 168
restore REPLACE 173, 193
SDRGetObjects 31
SDRRetrieveFile 37
selective backup 167
statvsd 139, 141
vdidl3 136
vdidl3mx 136
VSD commands 21
vsdatalst 38
Configuration Manager 19
configurations 199

D

Daemons
fault service daemon 28, 30
fault_service_Worm_RTG_SP 28
hags 36
mmfsd 36
data allocation map 18
Data partitioning 61
data throughput 199
dedicated 199
default parameters 93
Default sizing parameters 93
degraded mode 44
DFS 49
directories 18
directory blocks 18
disk/loop layout 99
disks 91, 93, 95, 96, 199
disks per node 99

E

ease of use 1
Example Sizing 94, 97

F

File System
Andrew File System 50
Comparison summary 51
DFS 49
JFS 45
NFS 46
PIOFS 48
File system capacity 88, 90
File System Manager 19
file system type 199
Files
act.top.pid 32
cluster.nodes 37
expected.top 31
filesystem 37
mmfs.cfg 37
mmsdrcfg1 37
mmsdrfs 37
topologies 31

G

General Parallel File System
Advantages 51
Availability 53
Block Size 57
block size parameter 175
Capacity 53
comparing ADSM performance with JFS 191
Comparison with other file systems 45

daemon resource requirements for ADSM implementation 171
File System Primitives 56
functional differences to JFS 166
inode size parameter 175
Limitations 54
mallocsize parameter 175
maxFilesToCache parameter 175
maximum file size 168
maximum file system size 168
Memory Mapped Files 55
metadata distributed over several SP nodes 168
Performance 52
Prefetch 61
read data flow 12
Reliability Issues 54
Scalability 51
Simplified Administration 54
Write Behind 61
write data flow 4
global access 1
GPFS
performance 86, 199
GPFS file system 199
GPFS Switch thoughput 99
Group Services 41

I

I/O contention 92
IBM Virtual Shared Disk 33
independent 64
indirect blocks 18
init_cache_buffer_count 94
initial configuration 93
initial parameters 94
inode allocation map 18
inodes 18

J

JBOD 88
Journalled File System 38

L

locking
advisory, POSIX standard 166
mandatory on JFS 166
Logging 41
loop bandwidth. 93

M

mallocsize 94
max file system bandwidth 87, 88, 97
Max GPFS File system block size 94
max_buddy_buffer_size 94
max_buddy_buffers 94
max_cache_buffer_count 94
max_coalesce 93
maximum aggregate bandwidth 87
maximum bandwidth through the SSA Adapter card 98
memory mapped files 166
Message Passing Interface 64
metadata 89, 199
Metadata Manager 20, 171
Metadata tests 199
min_buddy_buffer_size 94
minimum number of disks 90
Mirroring 43, 88
mmfsd 3, 17
mtime
delay of update 167
impact of delayed update on ADSM 167

N

Network File System 38
NFS 46
non full strided RAID-5 98
non-RAID 199
non-strided write 98
number and type of disks 97
number of adapter cards per node 93
number of servers 99
number of VSD Servers 90, 95

P

Pablo 69
pagepool 94
Parallel 199
performance 94, 96
performance per data disk 89
personalities 17
PIOFS 48
placement of Adapter 93

placement of SSA Adapters 93
portability 1
POWER3 98

Q
questions 199
quorum 19

R
RAID 43, 44
RAID-5 88, 97, 199
RAID-5 Array size tests 199
RAID-5 arrays 98
Random 199
random access 86
Read-Modify-Write 44
Recoverability 88, 95
reliability 2
Replication 43, 88
results 199
Round-Robin partitioning 61
rpoolsize 93
RVSD 41, 99
rw_request_count 94

S
scalable 1
Segmented partionining 61
sequential access 86
Serial 199
serialization 167
Sizing
  requirements 86
  steps 86
SP Switch 92, 93
  css device 27
  cssdd3 device driver 27
  fault service daemon 30
  fault_service_SP kernel extension 27
  good network performance between ADSM client and server 196
  if_ls interface layer 33
  initialization 30
  IP interface 33
    mbuf 34
    mcluster 34
    receive FIFO 34
    receive pool 34
    send FIFO 34
    send pool 34
  MX adapters 27
  PCI adapters 27
  Start up process 29
  throughput 92
  throughput rate 95
SSA adapter 90, 92, 93, 99
SSA adapter cards 91
SSA Adapter throughput 99
SSA Adapters 91, 92
SSA Disk Identification 90
SSA disks 96, 98
SSA loop bandwidth 93
SSA loop disk layout 96, 100
SSA loops 96, 199
stripe group 19
stripe group descriptor 18
Stripe Group Manager 19, 171
Subsystems
  RS/6000 Cluster Technology 35
    Event Management 37
    Group Services 35
    hags 36
    Network Connectivity Table 35
    Topology Services 35
  RSCT (see RS/6000 Cluster Technology)
  System Data Repository 37
    IBM Virtual Shared Disk 38
    mmfs.cfg 37
    mmsdrcfg1 37
    mmsdrfs 37
sustained throughput rate 89
Switch
  throughput 92

T
TCPIP
  rfc1323 parameter 175
  sb_max parameter 175
  tcp_mssdflt parameter 175
  tcp_recvspace parameter 175
  tcp_sendspace parameter 175
  thewall parameter 175
tests 199
thewall 127
time stamps of files 167

token 20
token management 89
Token Manager Server 20, 171
token stealing 20
total number of adapter cards 93
Tuning
  Applications 101
  buddy buffer 141
  Buddy Buffer considerations 105
  controlling a GPFS node 146
  Dedicating VSD servers 121
  disk activity problems 142
  Disk subsystem
    congested data path 144
    damaged disk 144
    Maximum Coalesce 143
    Queue Depth 143
  dropped packets 136
  failed requests 140
  File system block size 103
  File System parameters 133
  GPFS Configurations 101
  GPFS daemon priority 132
  High Impact issues 103
  IBM Virtual Shared Disk 135, 139
  IP interrupt queue 136, 138
  ipqmaxlen 123, 139
  Low impact issues 128
  mallocpool for ADSM performance 189
  mallocsize 125
  max_IP_msg_size 122
  maxFilesToCache 124, 191
  maxFileToCache for ADSM performance 189
  mbuf 138
  Medium impact issues 120
  metadata 121
  Network Parameters 101
  Number of GPFS nodes 123
  Number of IBM Virtual Shared Disks 104
  over-sized server environment 146
  Pagepool size 112
  pbuf considerations 107
  pbufs 141
  queue_depth 128
  receive pool 137
  rejected merge timeouts 141
  rejected requests 140, 141
  rejected responses 140
  Request block limit 110
  request blocks 141
  send pool 136
  statvsd 141
  Stripe group manager node 130
  switch 135, 136, 139
  switch adapter 135, 136
  switch congestion 145
  Switch receive pool and send pool settings 114
  The metadata manager node 133
  The striping algorithm 132
  thewall 138
  VSD cache 128
  VSD Configurations 101
  worker threads 146
  worker2Threads 126
tuning
  TCP/IP Parameters 127
twin-tailed 41
type of disks 89

U

ull strided write 91
uning 93

V

Virtual Shared Disk 20
  architecture 22
  client resource requirements for ADSM implementation 170
  command lines 21
  read data flow 23
  server resource requirements for ADSM implementation 170
  write data flow 23
VSD disks per Server 94
VSD Server configuration 93
VSD Servers 199
VSD servers 93, 199
vsd_max_ip_msg_size 94
vsd_request_count 94

ITSO Redbook evaluation

Sizing and Tuning GPFS
SG24-5610-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at `http://www.redbooks.ibm.com/`
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to `redbook@us.ibm.com`

Which of the following best describes you?
_ Customer _ Business Partner _ Solution Developer _ IBM employee
_ None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

What other redbooks would you like to see published?

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

SG24-5610-00
Printed in the U.S.A.

Sizing and Tuning GPFS

SG24-5610-00