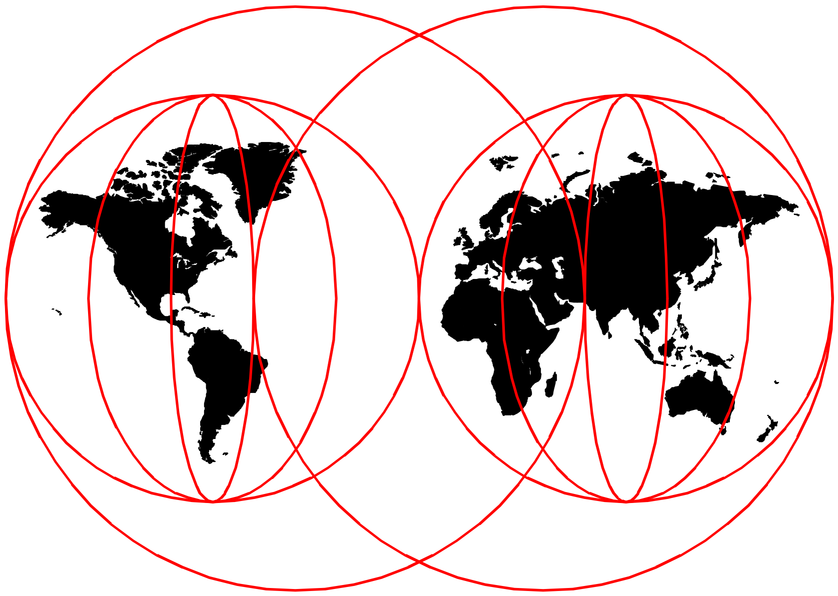
RS/6000 SP System Performance Tuning

Hajo Kitzhöfer, Andrew Dunshea, Frank Mogus



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-5340-00

SG24-5340-00

International Technical Support Organization

RS/6000 SP System Performance Tuning

May 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, “Special Notices” on page 279.

First Edition (May 1999)

This edition applies to Version 2, Release 4 and Version 3, Release 1 of the POWERparallel System Support Programs for use with AIX 4.3.1 and AIX 4.3.2

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Mail Station P099
522 South Road
Poughkeepsie, New York 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

Tables

Preface

The main tuning objectives for the RS/6000 SP are to improve performance, response time, and resource utilization. While the objectives of performance tuning for the RS/6000 SP are similar to those for a stand-alone RS/6000, due to its design and architecture, the approach is, in some situations, the opposite of how to tune a stand-alone system.

This redbook focuses on SP specific performance subjects. It is not meant as an update or replacement for the *AIX Performance Tuning Guide*, SR28-5930. This redbook tries to avoid overlaps with other performance redbooks.

This redbook gives some general system recommendations and describes specific tuning strategies for workloads, network interfaces, and file systems. It provides hints and tips on the various factors and variables that can enhance the performance of the system and applications and is essential documentation for people who support the RS/6000 SP.

Some knowledge of RS/6000, RS/6000 SP, and the AIX operating system is assumed.

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.

Dr Hajo Kitzhöfer is an Advisory International Technical Support Organization (ITSO) Specialist for RS/6000 SP at the Poughkeepsie Center. He holds a Ph.D. degree in Electrical Engineering from the Ruhr-University of Bochum (RUB). Before joining ITSO, he worked as an SP Specialist at the RS/6000 and AIX Competence Center, IBM Germany. He has worked at IBM for eight years. His areas of expertise include RS/6000 SP, SMP, and Benchmarks. He now specializes in SP System Management, SP Performance Tuning, and SP hardware.

Andrew Dunshea is a Performance Analyst from IBM New Zealand. He has 10 years of experience in application development. His areas of expertise include object-oriented software development and analysis, systems programming, and performance analysis.

Frank Mogus is a Systems Consultant from Canada. He has several years of UNIX experience and has worked with The Braegen Group for four years.

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

Bernard King-Smith
IBM PPS Lab Poughkeepsie

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 299 to the fax number shown on the form.
- Use the online evaluation form found at: `http://www.redbooks.ibm.com`
- Send us a note at the following address:

  `redbook@us.ibm.com`

Chapter 1. Introduction

This document is divided into three parts.

Part 1, "RS/6000 SP Basics," gives an overview of the SP hardware and software. Since the SP is essentially a collection of workstations connected by networks, tuning an SP system means primarily tuning these networks. Therefore, this part also includes an overview of the SP network topology and an introduction to TCP/IP.

Part 2, "SP System Performance Tuning," covers all tuning aspects of an SP. Readers with SP and TCP/IP knowledge might want to skip Part 1 and start Part 2.

Part 3, "Performance Tools," is a reference to AIX performance tools and gives some pointers to, and information about, useful non-IBM performance tools.

1.1 System Performance Tuning

As your SP system grows and gets more complex, performance monitoring and tuning becomes a key issue. It is an issue that you need to deal with on an ongoing basis through all the stages of your system's life cycle: From planning, to design, to testing, to production. The demand for performance never stops. Even after a system is in production, you need to keep monitoring and tuning and improving it as circumstances change. Performance tuning is vital.

New technologies demand new types of tuning. With client-server and distributed applications (mainly distributed databases in a commercial environment), your applications now communicate over a network instead of running on a single system.

As time goes by, more users use the system. The organization grows and more applications and data are added. As users, data, and capabilities grow, your system needs to get tuned so it can absorb the increased load and continue to give the performance that is needed.

1.2 Why Tuning?

Tuning saves your organization money in several ways:

- A high-performance, well-tuned system produces faster response times. This makes users more productive.
- A well-tuned system can delay the necessity of buying additional equipment.

In addition to the financial benefits of performance tuning, there are human benefits to consider. Nothing can be more frustrating for an employee who is trying to be productive than having to wait for computing resources or dealing with painfully slow response times.

1.3 When to Tune?

Too many people think of tuning as an add-on. They start tinkering when users start complaining. That is not the proper approach.

Performance is a design goal, something to build into the system from the beginning, not something that will be done when things go wrong, and time is short. Also, performance tuning is never finished.

1.4 How Much Tuning Is Enough?

When do you know that you have tuned enough? Experience shows that many sites tune too little, but that a few actually tune too much. Be sure not to make too many changes at once because you will not know which of the changes improved performance.

There comes a point when the system is in balance, and it is better not to fiddle around with the settings when trying to achieve infinitesimally small performance improvements.

Sometimes, improving performance in one area of the system degrades performance in another. For example, tuning the switch network for optimal performance for a parallel database may actually hurt the performance of client-server applications, such as ADSM running on the same nodes.

You have to be careful about making changes once your system is in production. For best results, always try to build a test environment and monitor performance in this environment before making any changes to the production system.

Part 1. RS/6000 SP Basics

Chapter 2. RS/6000 SP Overview

Before we start tuning the RS/6000 SP, we should know the base elements of the system. This chapter gives an overview of the RS/6000 SP hardware and software components.

2.1 SP Topology Overview

In December 1991, IBM built a second family of parallel products based on the RS/6000 technology (the other is the System/390 line of parallel servers). People inside and outside IBM were connecting their RS/6000 systems with local area networks into clusters and using them in place of much more expensive supercomputers. So, IBM chose this cluster approach as the basis for development. The basic design objectives for the RS/6000 Scalable POWERparallel System (SP) were:

General-purpose machine	To be capable of handling a variety of workloads and applications, not limited to parallel computing, but also useful as a server consolidation platform.
Common components	To use common components, such as POWER2 processors, the AIX operating system, standard networks, and others.
Fast interconnection	To be a highly flexible and elegant switch network for high-performance non-blocking interconnections.

Because of these design goals, the SP is:

- Scalable

  Being based on standard technology, the system provides incremental growth for all resources including the number of processing nodes and associated memory, disks, I/O, as well as switch bandwidth.

- Upgradable

  Incorporating new technology in existing systems provides a growth path without requiring a complete system replacement. Examples of new technology are processor nodes, disks, and the switch, each of which can be replaced independent of the other components.

- Highly Available

The system design incorporates high-availability characteristics, such as redundant components, error detection and correction circuitry, and advanced diagnostics.

- Manageable

  Features, such as single point-of-control and hardware and system monitoring, are used to maintain system administrator productivity.

The difference between a stand-alone RS/6000 and the RS/6000 SP is the capability of the RS/6000 SP to perform as a parallel architecture machine. It is a distributed memory computer, that is, all nodes have their own memory as well as other I/O resources. The main component that can be shared in this architecture is the SP Switch, which has the capability of transferring data at more than 150 MBps per node. The RS/6000 SP is centrally managed from a Control Workstation (CWS) with the Parallel Systems Support Program (PSSP). Figure 1 shows a typical RS/6000 SP environment.

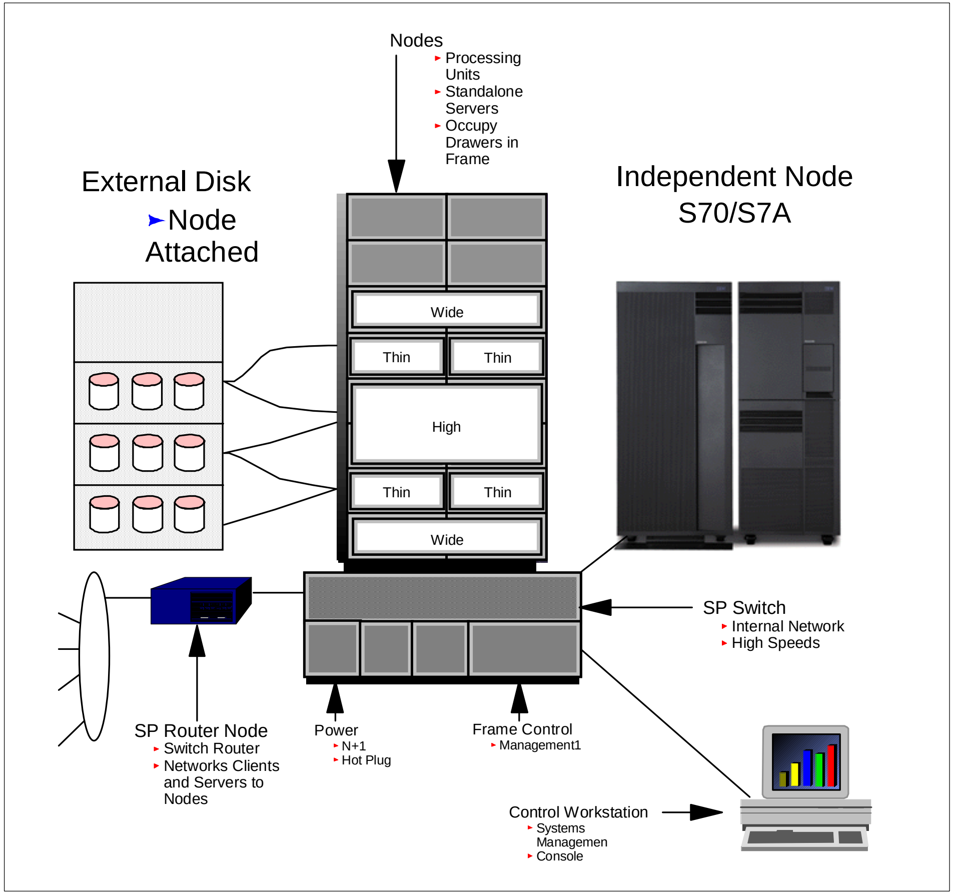


Figure 1. Typical RS/6000 SP Environment

2.1.1 SP Hardware

This section describes the basic hardware components of the RS/6000 SP.

Frames

SP processor nodes are mounted in a frame. There are two types of frames: A tall frame and a short (low-cost) frame. The tall frame consists of a redundant power supply, while the low-cost frame has one power supply with the option of adding a second one. The frames have drawers where the nodes reside. A tall frame has the capability to hold eight drawers, while the low-cost frame can hold four drawers. Each drawer can be further subdivided into two slots. Below the bottom-most drawer in the frame is where the optional SP Switch board resides. The switch board does not occupy node drawer space.

The frames communicate serially with the CWS through a frame supervisor card. The frames can be interconnected to form a system of up to 512 nodes. As soon as the number of nodes exceeds 80, additional switch boards, referred to as Intermediate Switch Boards (ISBs), are cascaded to the Node Switch boards effectively adding one more stage to the Switch network. For more details, see Figure 8 on page 19. The ISBs are housed in a separate frame; the total number of ISBs placed in this separate frame is eight. For more information, see 3.2, “The Switch Network” on page 15.

Nodes

The basic building block of the RS/6000 SP is the processor node. These nodes are usually housed in the frame, and, therefore, are called *internal nodes*.

There are three different types of internal nodes:

- Thin Node - Uniprocessor or SMP occupies one slot (two per drawer).
- Wide Node - Uniprocessor or SMP occupies two slots or one drawer.
- High Node - SMP occupies two drawers.

In addition to the internal nodes, two kinds of external nodes are supported: The dependent node and the independent extension node.

The RS/6000 SP Switch Router is currently the only incarnation of a dependent node. This node combines the Ascend GRF with IBM RS/6000 SP Switch Router Adapter to enable a fast direct network attachment to the RS/6000 SP Switch (see Figure 1 on page 7).

An example of external independent nodes are the S70/S7A servers; these 64-bit SMP systems can be integrated into an RS/6000 SP (see Figure 1 on page 7) using a PCI Switch Adapter for the connection to the SP Switch network.

Control Workstation

The CWS is used to control and monitor an RS/6000 SP environment. The beauty of the RS/6000 SP is that it has this single point of control. It connects to the frame through RS-232 lines to perform hardware monitoring of the nodes and the frame itself. Each frame must have an RS-232 connection to the CWS for diagnostics, maintenance, and console access to the nodes.

The Switch

The SP Switch is an optional element of the RS/6000 SP. It is a high-performance interconnect between all the nodes. The total number of nodes that can be connected with one switch board is 16. Once these 16 nodes are connected, a second switch is needed. The switch has a non-blocking nature, meaning that unlike broadcast networks, where there can only be one device on the network sending or receiving data, the switch has up to four paths. And, in fact, the more paths that are used, the more efficient the switch is. An Intermediate Switch Board is needed once you have filled five full frames or up to 80 nodes. More information on the switch and networks is covered in Chapter 3, “Network Topology” on page 13.

More details about the SP hardware and some information about selection criteria for the nodes can be found in Appendix B, “Hardware Details” on page 255 or in *Inside the RS/6000 SP,* SG24-5145.

2.1.2 SP Software

This section gives an overview of the software components used by an RS/6000 SP.

AIX

Advanced Interactive Executive (AIX) is IBM’s version of Unix. It is the operating system on which the hardware executes.

PSSP

Parallel Systems Software Program (PSSP) is the software that installs, configures, and controls the nodes. The nodes are centrally managed from the CWS. PSSP uses Network Installation Management (NIM) to install, maintain, or boot the nodes from the CWS. The hardware control and monitoring part of the PSSP software communicates with the frame through the Frame Supervisor Card (FSC). The communication uses an RS-232 serial line connection. The Frame Supervisor is connected to the Node Supervisor Card (NSC) and also to the Switch Supervisor (which is part of the switchboard).

2.2 Communication Paths

Two communication paths between the nodes and the CWS (SP Ethernet network) and between the frames and the CWS (RS 232) are mandatory for an SP system. The switch network is optional.

• RS232 Hardware Monitoring Line

The mandatory RS232 hardware monitoring line connects the CWS to each RS/6000 SP frame primarily used for node and frame hardware monitoring.

- SP Ethernet

  One of the prerequisites of the RS/6000 SP is an internal BNC or 10BaseT network. The purpose of this network is to install the nodes' operating systems and the PSSP software and also to diagnose and maintain the RS/6000 SP complex through the PSSP software.

2.3 System Partitioning

The RS/6000 SP enables you to divide the system into logically separate systems. This concept of system partitioning is very similar to the Virtual Machine in the mainframe environment, which supports different machine images running in parallel; you can have a production image and a test image within the same machine. The RS/6000 SP provides almost complete isolated environments within the complex.

2.4 Homogenous versus Consolidated System

A homogenous environment in the RS/6000 SP environment is a cluster of nodes; all of which perform similar tasks. An example of a homogenous environment would be all nodes that are parallel database servers. A request would come in to one node, and that node would dispatch the request to the other nodes that make up the specific cluster.

The main characteristic of a homogenous system is the ease with which it can be tuned as all nodes perform the same task. Once the first node is tuned, the changes can be propagated to all others.

A consolidated system consists of various types of nodes performing distinct roles. An example of a consolidated environment would be to have a few nodes dedicated to a classical database environment (where no parallel processing exists) along with a couple of nodes that would house Lotus Notes servers as well as a dedicated ADSM backup node.

The main characteristic of consolidated nodes is that they consist of many nodes performing a variety of tasks. Consolidated systems present some of the most complex challenges when it comes to performance tuning.

2.5 Logical versus Physical View

When providing any performance details on an RS/6000 SP system, it is important to group the system into logical classes. These classes are groups of nodes with similar characteristics; as an example, see Figure 2.

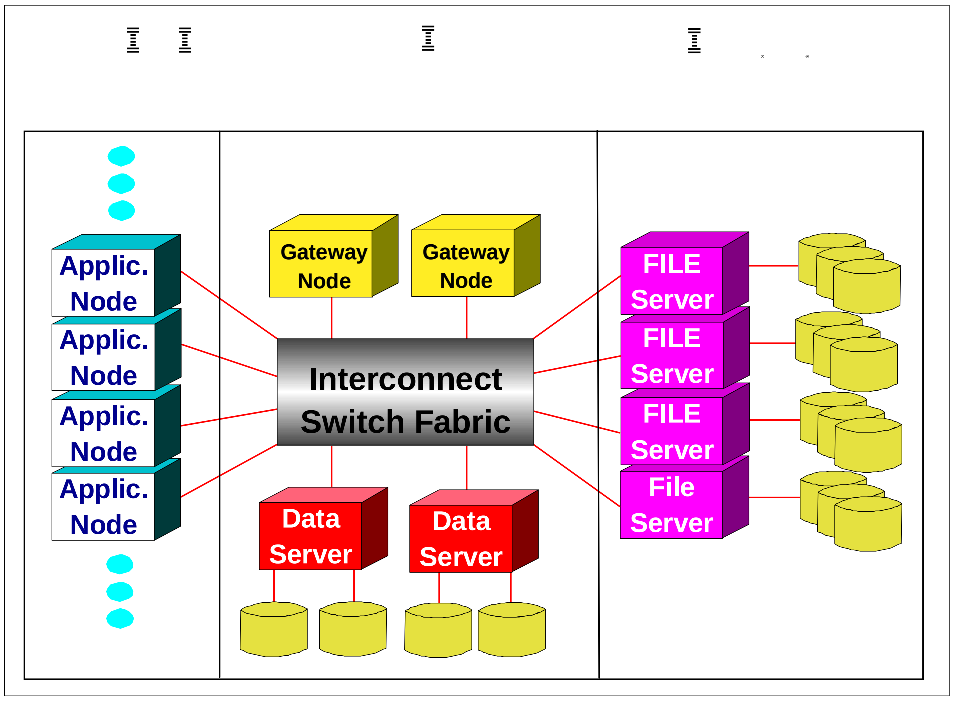


Figure 2. Grouping Nodes by Classes of Service

The best way to achieve this goal is to document your system in two steps. First, take a physical inventory of your RS/6000 SP environment including all peripheral devices. Next, take an inventory of the types of processing on the nodes. Is this a homogenous or consolidated environment? If homogenous, then this exercise becomes redundant after a few nodes, but it must be finished before we can depict the complete environment. If consolidated, then the exercise becomes even more crucial to gain a precise understanding of the system. Once the environment is organized into types of applications per node, we can draw a logical diagram as depicted in Figure 2 on page 11.

Chapter 3. Network Topology

SP networks consist of SP Ethernet, RS 232 connections between CWS and frames, and the optional SP Switch network.

The choice of topology is one factor in achieving both scalability and modularity. A variety of topologies have been chosen for connecting existing commercial parallel systems.

Bus-based systems are well suited to connect a small number of nodes but are limited by a total bus bandwidth that does not increase as more processors are added. They are, therefore, not appropriate for connecting hundreds of nodes.

To overcome bus scalability problems, massive parallel processing (MPP) uses point-to-point interconnection networks. These networks are constructed by connecting switch elements by point-to-point links.

3.1 The Ethernet Network

The SP Ethernet Network connects all the nodes to the CWS. The SP Ethernet is used to monitor, install, update, boot, and customize the nodes and should be dedicated to SP operating traffic. See a sample network configuration in Figure 3 on page 14.

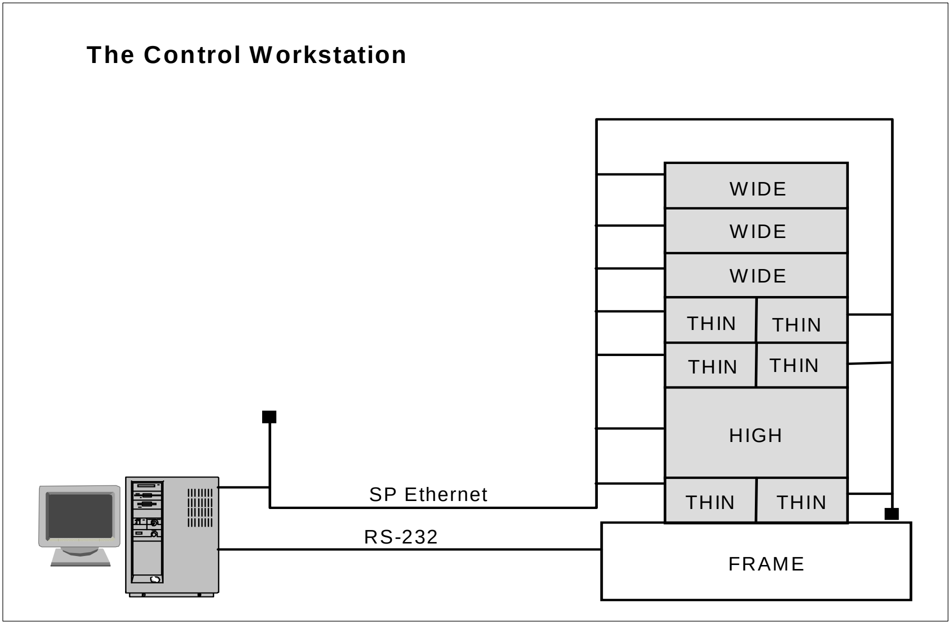


Figure 3. One Frame Ethernet Configuration

Scalability of the RS/6000 SP Ethernet network becomes increasingly important as the number of nodes increases. The Ethernet can scale up to 30 participants per physical segment. If more than 29 nodes are used, another node must be set up as gateway or router to connect different networks. The use of repeaters, bridges, or an Ethernet switch may be an alternative. Figure 4 on page 15 shows a sample of large SP configuration.

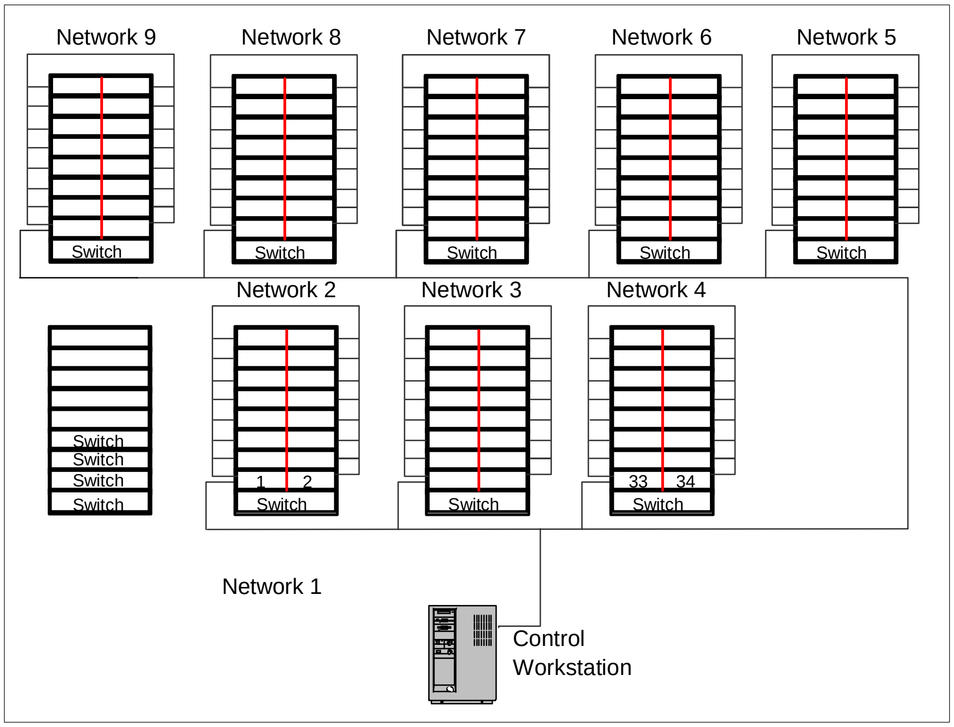


Figure 4. 8-Frame Ethernet Configuration

When performing a network installation on more than eight nodes, it is necessary to build a tree structure with the network so that several levels of installation nodes exist. This method allows many nodes to be installed at once while limiting the installation to eight nodes per physical subnet or network.

3.2 The Switch Network

The dominant goals for the SP communication subsystem are scalability, modularity, and ease of integration with the processing nodes. The objective for *scalability* is a network that increases its aggregate bandwidth linearly as the number of nodes increases while maintaining low average latency for message transfer.

The goal of *modularity* is to provide cost-effective networks for small systems that function as building blocks for large systems. Finally, we require the ability to quickly integrate the latest processor technology.

Switching elements are devices with multiple input and output ports that forward packets arriving at an input port to a desired output port. These switch elements are non-blocking; that is, if a packet arriving at an input port x is destined for a particular output port y, and no other received packets are destined for y, then this packet may immediately be forwarded to y, regardless of other received packets.

The switching element is also called the switch chip. Each such chip contains eight switch ports and a crossbar that allows packets to pass directly from port to port. These crossbar paths allow packets to pass through the switch with low latency. Figure 5 shows a switch chip with eight link interfaces (each link interface comprises two ports, one input, and one output port) and the crossbar.

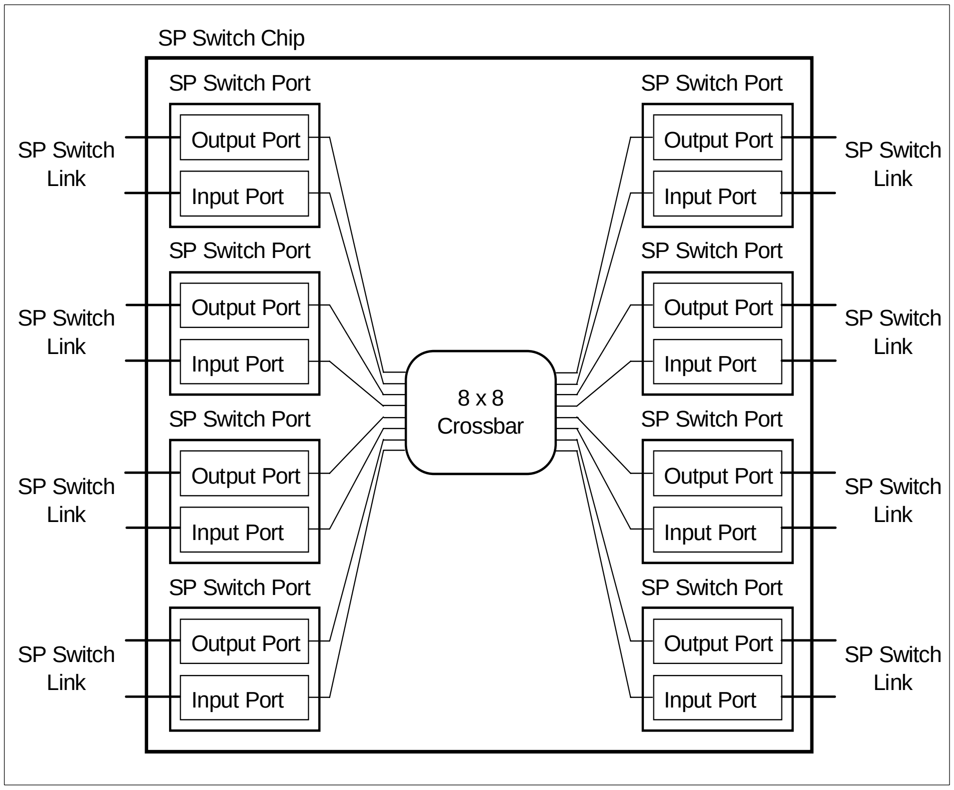


Figure 5. SP Switch Chip

An SP Switch board contains two fully interconnected columns, or stages, of four switch chips as shown in Figure 6 on page 17. Note that there is a path between any two external links.

These boards are building blocks used in constructing larger SP networks.

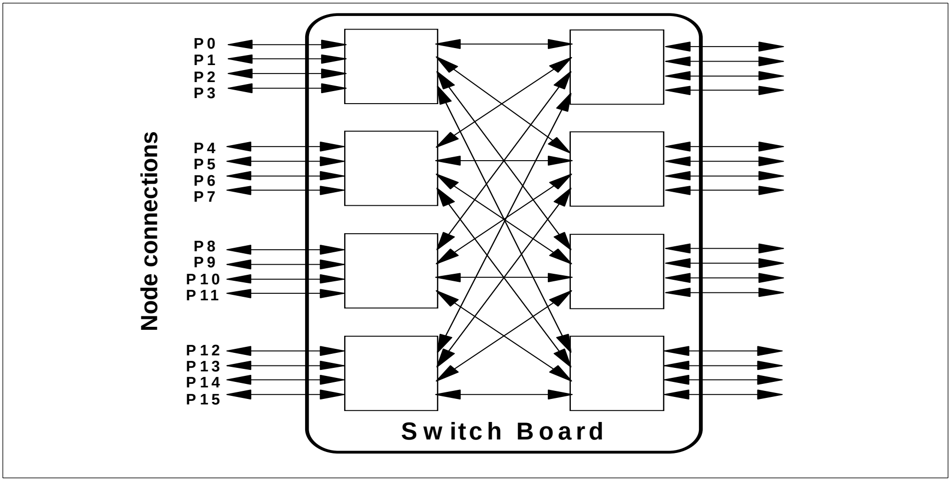


Figure 6. SP Switch Board

SP nodes are grouped into 16-processor units that are connected to one side of the switch boards (also called node switch boards because of their direct connection to the nodes).

The 16 unused links on the right side of the node switch board are used for creating larger networks in one of two ways:

- For systems containing up to 80 nodes, these links connect directly to the right sides of the other node switch boards. See Figure 7 on page 18 for an example of such a network configuration.

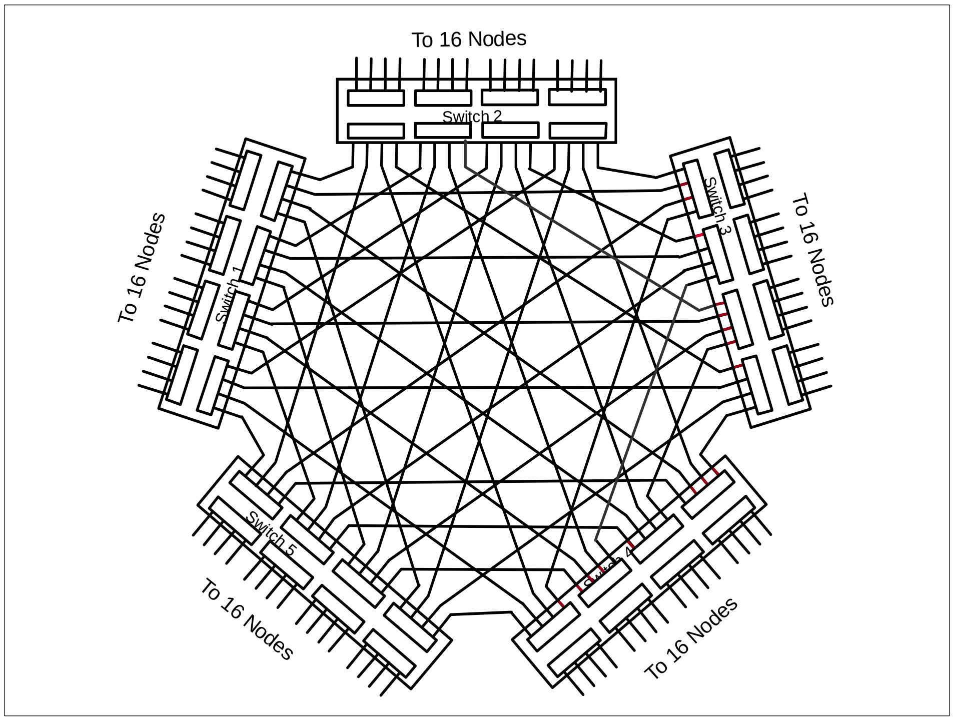


Figure 7. SP 80-way System

- As soon as you have more than 80 nodes, it gets a little bit more complicated. Now, the links are connected to additional stages of switch boards. See Figure 8 on page 19 for a switch network with more than 80 nodes.

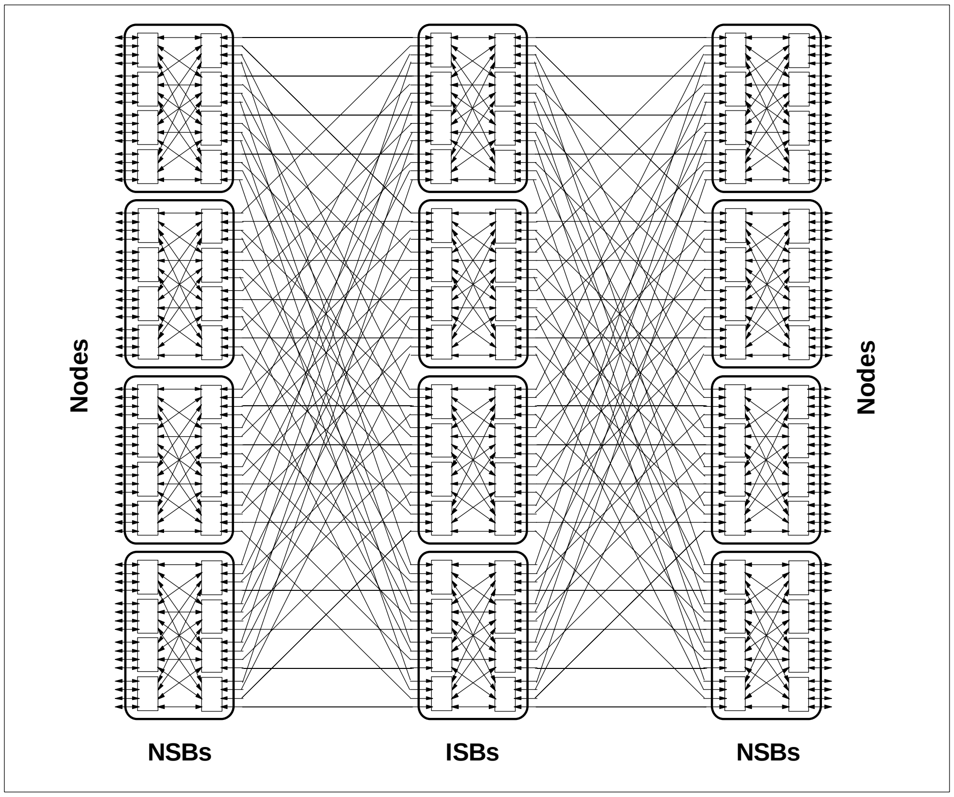


Figure 8. SP Switch Network with More Than 80 Nodes

Additional switch boards are added to the network topology. Since these boards are not directly connected to nodes, they are called *intermediate switch boards* (ISBs), while the boards that are directly connected to nodes are called *node switch boards* (NSB).

Each node is connected to the switch board through a switch adapter. Currently, there are three different types of switch adapters available. Each adapter has two ports connected to the switch fabric: One input and one output port (see Figure 9 on page 20). Even though there are two separate ports, only one cable connects the adapter to the switch board.

The main difference between these adapters is their internal connection within the nodes. Two adapters are standard bus-attached adapters, one of which is designed for the Micro Channel Architecture (MCA), the other for the PCI Bus system (currently only supported for the S70 or S7A nodes). The third type of adapter connects to the MX-Bus within newer PCI-based nodes.

MX-Bus stands for mezzanine-extended-Bus and is a direct 1-slot connection to the internal memory bus within PCI nodes.

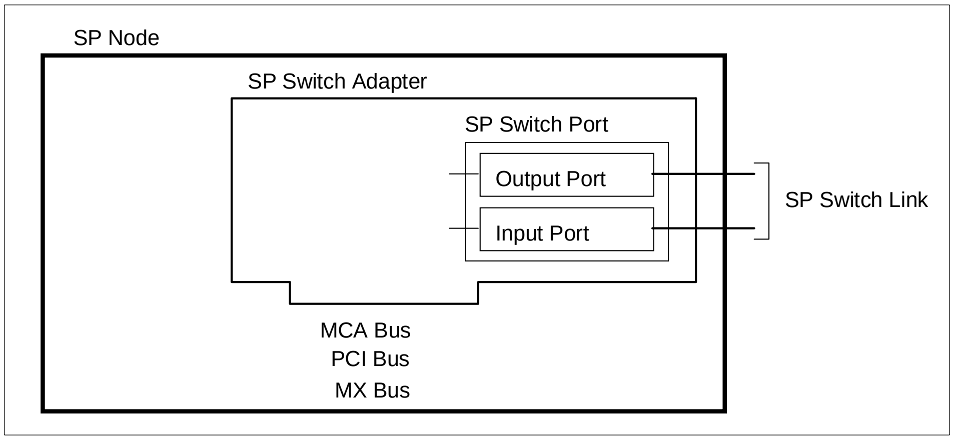


Figure 9. Switch Adapter

More details about the SP switch can be found in the following documents: *Understanding and Using the SP Switch,* SG24-5161, *Inside the RS/6000 SP,* SG24-5145, and *IBM System Journal, Vol 34, No. 2,* 1995.

Chapter 4. TCP/IP Overview

The SP is essentially a network connected collection of workstations. Because the communication network is the most import subject for any tuning considerations for an SP, we should get a general understanding of how the different layers of the TCP/IP protocol stack interact.

TCP/IP consists of several communication layers. There are parameters that impact the different protocol layers. They can best be understood by breaking them down into categories:

- The no parameters are the initial network options that affect TCP, UDP, and IP and are independent to the adapter type.
- The MTU, or maximum transmission unit, is the largest possible packet size that can be sent on a specific physical medium (Ethernet, Token-Ring, SP Switch, and so on).
- The adapter queues specify the number of packets that can be queued on a specific adapter while it is sending or receiving data. These are specific to an adapter even if there are other adapters of the same type.
- The SP Switch uses switch pools instead. These switch adapter pools define the amount of pinned kernel memory to be used by the switch device driver to handle network traffic destined for the SP Switch.

For a review of the TCP/IP layer model and to clarify the interrelationships, let's break this down further, step by step:

1. An application performs a write request. Data is copied from the application's working buffer to the socket send buffer.
2. The socket layer passes the data to TCP or UDP.
3. For remote networks, if the data is larger than the maximum segment size (MSS), TCP breaks the data into fragments that comply with the MSS.
4. For local networks, if the data is larger than the MTU, TCP breaks the data into fragments that comply with the MTU.
5. UDP leaves the fragmentation to the IP layer.
6. The Interface Layer makes sure that no packet exceeds the MTU.
7. The packets are then placed on the adapter output queue or the SP Switch sendpool and are transmitted to the receiving system.

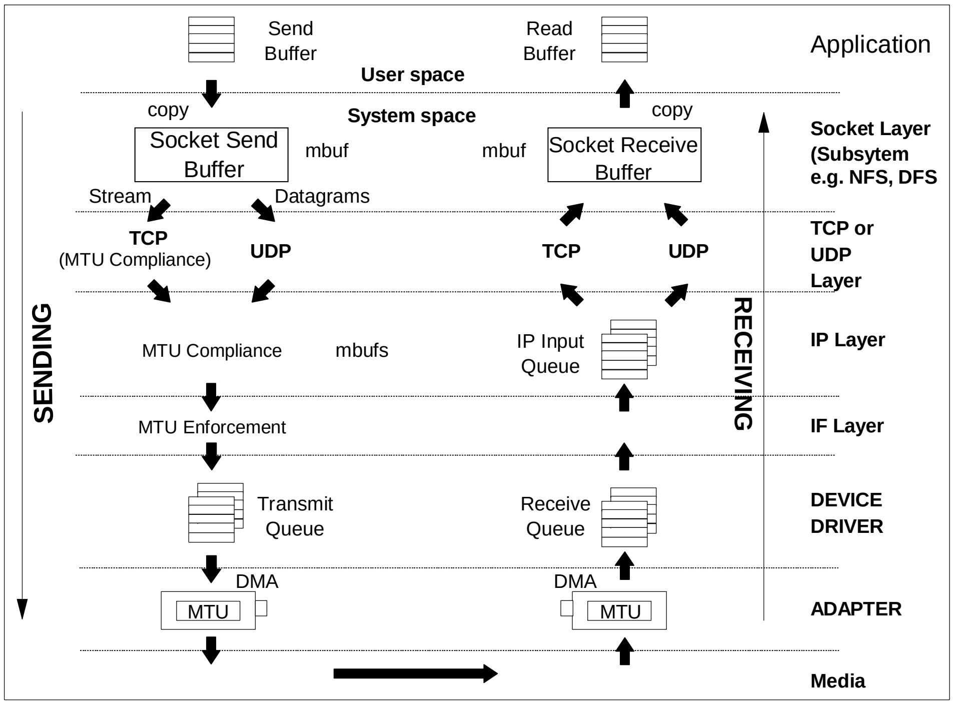


Figure 10. TCP/UDP/IP Data Flow

8. The receiving host places the incoming packets on the adapter's receive queue. They are then passed up to the IP layer.
9. The IP layer then determines if any fragmentation has taken place due to the MTU. If so, it puts the fragments back to their original form and passes the packets to TCP or UDP.
10. TCP reassembles the original segments and puts them on the socket receive buffer in kernel memory, or UDP passes the data on to the socket receive buffer in kernel memory.
11. The application's read request causes the appropriate data to be copied from the socket receive buffer to the buffer in the application's working area.

There are many parameters that affect your network performance. At the device driver layer, you have your transmit queue size marked by the parameters xmt_que_size or spoolsize. You also have your receive queue size marked by rec_que_size or rpoolsize.

At the interface layer, you have enforcement of the MTU or segment size as it pertains to what type of network media is being used: Ethernet, Token-Ring, SP Switch, or others.

At the transport layer, your performance parameters are set by tcp/udp send/recvspace.

You also have the socket layer between the transport and application layers, the parameter sb_max, which determines the maximum amount of memory or mbuf space that can be used by TCP or UDP for socket buffers for each socket.

Lastly, the parameters listed above all impact system memory. The thewall parameter determines the maximum amount of buffer space that can be used across the entire communication subsystems.

4.1 Communication Subsystem Memory Management

The design of the communication subsystem memory management changed with the latest incarnations of AIX. One major requirement for the algorithm is to efficiently allocate and reclaim pinned (physical) memory within the communication subsystem. This is achieved through the use of mbufs and additional buffers called clusters. An mbuf is a 256-byte data buffer. Prior to AIX 4.2.1, a mcluster had a fixed size of 4096 bytes, or one page of memory. That is, you could only use a combination of mbufs and mclusters allocated for the data.

It is up to the device driver to inform TCP or UDP what size of mcluster to allocate. They use multiple buffers, each of the same data size.

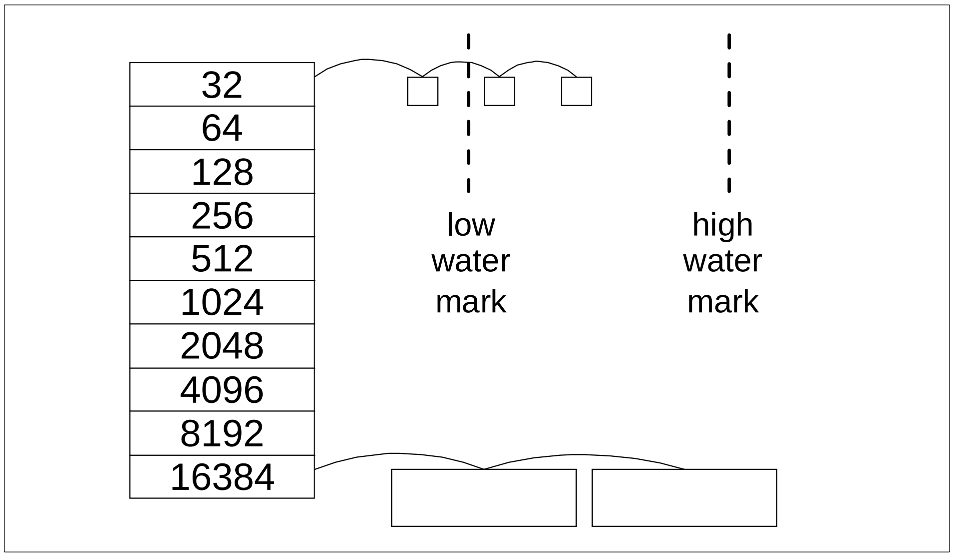


Figure 11. Cluster Chains

This scheme reduces the amount of pinned memory by not preallocating a set of mbuf buffers and allows the system to give memory to the buckets currently in demand. Each bucket keeps its own high and low watermark. Smaller buckets can steal from larger buckets. This ability allows this scheme to be self-tuning.

Use netstat -m to get an overview of cluster usage and to which subsystems clusters are allocated. See Figure 12 on page 25 for an overview.

```
Kernel malloc statistics:

******* CPU 0 *******
By size       inuse     calls failed    free   hiwat   freed
32              135     17392      0     121     640       0
64               77      1826      0      51     320       0
128              52      2678      0      12     160       0
256              42   3129745      0      86     384       0
512              51      3594      0      29      40       3
1024             22     19845      0      62     100       0
2048              0       768      0       4     100       0
4096              2       664      0       5     120       0
16384             1      1026      0      18      24       7
32768             1         1      0       0    2048       0

******* CPU 1 *******
By size       inuse     calls failed    free   hiwat   freed
32               19     12593      0     109     640       0
64               14      2522      0      50     320       0
128               6      2345      0      26     160       0
256              59   2980466      0      69     384       0
512              23      3946      0      41      40       0
1024              3     15438      0      65     100       0
2048              0       716      0       6     100       0
4096              1       641      0       1     120       0
16384             0       916      0      18      24       8

******* CPU 2 *******
By size       inuse     calls failed    free   hiwat   freed
32                9      2903      0     119     640       0
...

By type       inuse     calls failed  memuse  memmax  mapb

Streams mblk statistic failures:
0 high priority mblk failures
0 medium priority mblk failures
0 low priority mblk failures
```

Figure 12. netstat -m Output

As you can see from the command output, each CPU in an SMP system has a dedicated network memory pool. This is to improve performance by having pools dedicated per CPU. This eliminates the need for looking on mbuf pool access. In this screen, we have shortened the output from an 8-way high node.

By setting the extendednetstat parameter with the `no` command, you will get more detailed information. For example:

```
no -o xtendednetstat=1
```

Figure 13 shows the output of the `netstat -m` command after the modification. In this screen, you will notice that the `netstat` command now gives more information about the use of the various buffers.

```
Kernel malloc statistics:

******* CPU 0 *******
By size       inuse     calls failed    free   hiwat   freed
32              135     17394      0     121     640       0
64               77      1827      0      51     320       0
128              53      2760      0      11     160       0
256              42   3130406      0      86     384       0
512              51      3596      0      29      40       3
1024             22     19847      0      62     100       0
2048              0       768      0       4     100       0
4096              2       664      0       5     120       0
16384             1      1026      0      18      24       7
32768             1         1      0       0    2048       0

******* CPU 1 *******
By size       inuse     calls failed    free   hiwat   freed
32               19     12598      0     109     640       0
64               14      2526      0      50     320       0
...

By type       inuse     calls failed  memuse  memmax  mapb
mbuf             21       452      0    5376   10240     0
socket          321         8      0    1348    1120     0
pcb             728         4      0      78     128     0
fragtbl           0         4      0       0      32     0
...

Streams mblk statistic failures:
0 high priority mblk failures
0 medium priority mblk failures
0 low priority mblk failures
```

Figure 13. netstat -m with extendednetstat=1

The `inuse` column shows how many pinned pieces of kernel virtual memory are currently used, which means that they always reside in physical memory and are never paged out.

In addition to avoiding duplication, sharing the mbuf and cluster pools allows the various layers to pass pointers to one another reducing mbuf management calls and copying of data.

Starting with AIX 4.1, the only network option used to tune the network maximum memory pool is thewall. Since the new scheme is self-tuning, there is no need to tune any other parameter. If the system memory requirements exceed thewall, it will start to drop packets.

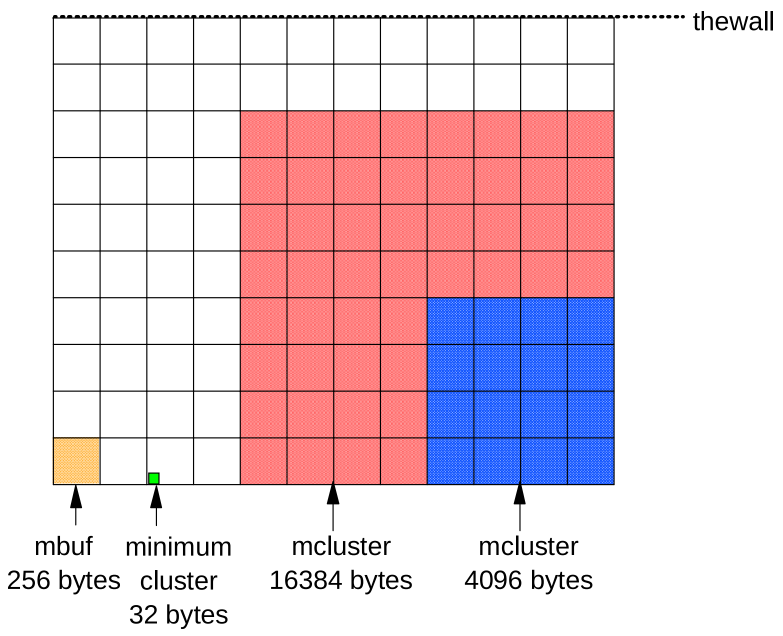


Figure 14. The Network Memory Pool

4.2 Socket Layer

Sockets provide the application program interface (API) to the communication subsystem. There are several types of sockets that provide various levels of service by using different communication protocols. Sockets of type SOCK_DGRAM use the UDP protocol. Sockets of type SOCK_STREAM use the TCP protocol. See Figure 15 on page 29 for an overview.

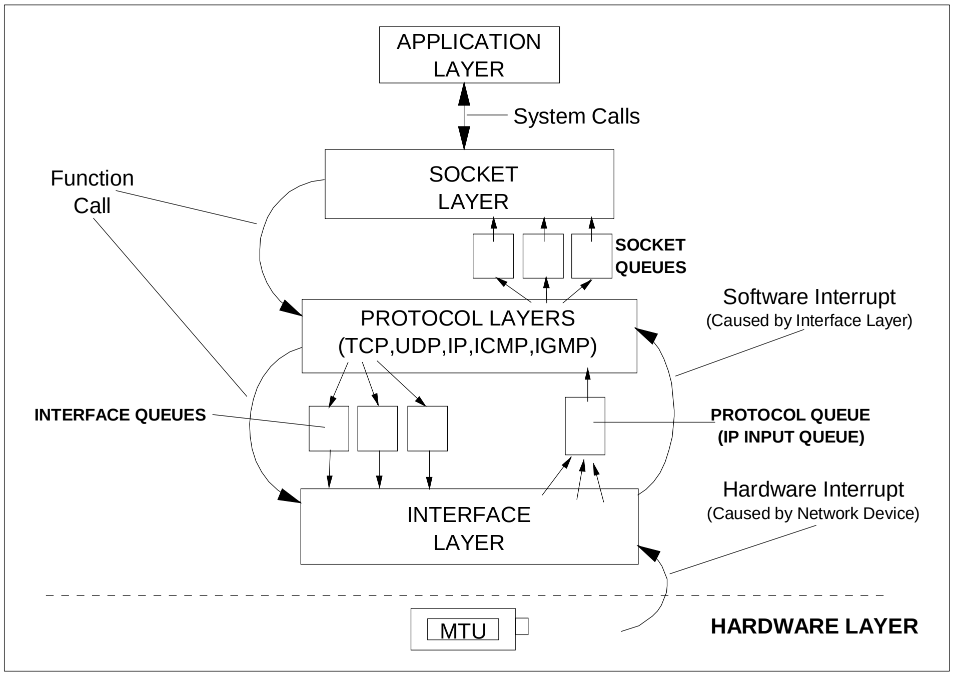


Figure 15. Socket Layer

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

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

- udp_sendspace and udp_recvspace

  The buffer sizes per datagram socket. The SP defaults recommendations are 32768 and 65536.

- tcp_sendspace and tcp_recvspace

  The buffer sizes per stream socket. The SP default for both values is 65536.

Note

Socket send or receive buffer sizes are limited to no more than *sb_max* bytes because *sb_max* is a ceiling on buffer space consumption. However, the two quantities are not measured in the same way.

The socket buffer size limits the number of bytes that can be held in the socket buffers. *sb_max* limits the amount of space in buffers of network memory pool that can allocated to a socket at any given time.

Send Flow

As an application writes to a socket, the data is copied from user space into the socket send buffer in kernel space. Depending on the amount of data being copied into the socket Data size send buffer, the socket puts the data into either mbufs or mclusters. Once the data is copied into the socket send buffer, the socket layer calls the transport layer (either TCP or UDP) passing it a pointer to the linked list of mbufs (an mbuf chain) and mclusters.

Receive Flow

On the receive side, an application opens a socket and attempts to read data from it. If there is no data in the socket receive buffer, the socket layer causes the application thread to go to the sleep state (blocking) until data arrives. When data arrives, it is put on the receive socket buffer queue and the application thread is made dispatchable. The data is then copied into the application’s buffer in user space, the mbuf chain is freed, and control is returned to the application.

4.3 UDP and TCP Functions

The following two sections describe the functions of UDP and TCP. To facilitate a comparison of UDP and TCP, the descriptions are divided into the following subsections:

- Connection
- Error detection
- Error recovery
- Flow control
- Data size
- MTU handling

4.3.1 UDP Layer

UDP provides a low-cost protocol for applications that have the facilities to deal with communication failures. UDP is most suitable for *request-response* applications. Since such an application has to handle a failure to respond anyway, it is little additional effort to handle communication errors as one of the causes of failure to respond to. For this reason, and because of its low overhead, subsystems such as NFS, ONC RPC, DCE RPC, and DFS use UDP.

Table 1. UDP Functions

Function	Attribute	Description
Connection	None	UDP is essentially a stateless protocol. Each request received from the caller is handled independently of those that precede or follow it.
Error detection	Checksum creation and verification	The sending UDP builds the checksum and the receiving UDP checks it. If the check fails, the packet is dropped.
Error recovery	None	UDP does not acknowledge receipt of packets, nor does it detect their loss in transmission or through buffer-pool overflow. Consequently, UDP never retransmits a packet. Recovery must be performed by the application.
Flow control	None	When UDP is asked to send, it sends the packet to IP. When a packet arrives from IP, it is placed in the socket-receive buffer. If either the device driver/adapter buffer queue or the socket-receive buffer is full when the packet arrives there, the packet is dropped without an error indication. The application or subsystem that sent the packet must detect the failure by timeout and retry the transmission. Also, the application must detect out-of-order packets and reorder them if necessary.
Data size	Must fit in one buffer	This means that the buffer pools on both sides of UDP must have buffer sizes that are adequate for an application’s requirements. The maximum size of a UDP packet is 64 KB. Of course, an application that builds large blocks can break them into multiple datagrams itself (DCE is an example) but it is simpler to use TCP.
MTU handling	None	Dealing with data larger than the maximum transfer unit (MTU) size for the interface is left to IP. If IP has to fragment the data to make it fit the MTU, loss of one of the fragments becomes an error that the application or subsystem must deal with.

Send Flow

If udp_sendspace is large enough to hold the datagram, the application’s data is copied into mbufs in kernel memory. If the datagram is larger than udp_sendspace, an error is returned to the application.

Receive Flow

UDP verifies the checksum and queues the data onto the proper socket. If the udp_recvspace limit is exceeded, the packet is discarded. If the application is waiting on a receive or read on the socket, it is put on the run queue. This causes the receive to copy the datagram into the user’s address space and release the mbufs; the receive is complete. Normally, the receiver will respond to the sender to acknowledge the receipt and to return a response message.

4.3.2 TCP Layer

TCP provides a reliable transmission protocol. With TCP ensuring that packets reach their destination, the application is freed from error detection and recovery responsibilities. Applications that use TCP transport include ftp, rcp, and Telnet. DCE and NFS can use TCP if it is configured to use a connection-oriented protocol.

Table 2. TCP Functions

Function	Attribute	Description
Connection	Explicit	The instance of TCP that receives the connection request from an application (we will call it the initiator) establishes a session with its counterpart on the other system, which we will call the listener. All exchanges of data and control packets are within the context of that session.
Error detection	Checksum creation and verification	The sending TCP side builds the checksum and the receiving TCP side checks it. If checksum verification fails, the receiver does not acknowledge receipt of the packet.
Error recovery	Full	TCP detects checksum failures and loss of a packet or fragment through timeout. In error situations, TCP retransmits the data until it is received correctly (or notifies the application of an unrecoverable error).
Flow control	Enforced	TCP uses a discipline called a sliding window to ensure delivery to the receiving application. The sliding window concept is illustrated in Figure 22 on page 70. (The records shown in the figure are for clarity only. TCP processes data as a stream of bytes; it does not keep track of record boundaries, which are application-defined.)
Data size	Indefinite	TCP does not process records or blocks; it processes a stream of bytes. If a send buffer is larger than the receiver can handle, it is segmented into MTU-sized packets.
MTU handling	Handled by segmentation in TCP	When the connection is established, the initiator and the listener negotiate a maximum segment size (MSS) to be used. The MSS is normally smaller than the MTU. If the output packet size exceeds the MSS, TCP does the segmentation, thus making fragmentation in IP unnecessary. See also “Maximum Segment Size (MSS)” on page 67.

Send Flow

When the TCP layer receives a write request from the socket layer, it allocates a new mbuf for its header information and copies the data in the socket-send buffer either into the TCP-header mbuf, if there is room, or into a newly allocated mbuf chain. TCP then checksums the data, updates its various state variables, which are used for flow control and other services, and finally calls the IP layer with the header mbuf now linked to the new mbuf chain.

Receive Flow

When the TCP input routine receives input data from IP, it checksums the TCP header and data for corruption detection, determines which connection this data is for, removes its header information, links the mbuf chain onto the socket-receive buffer associated with this connection, and uses a socket service to wake up the application (if it is sleeping as described earlier).

4.4 Internet Protocol Layer

The Internet Protocol (IP) provides a basic datagram service to the higher layers. If it is given a packet larger than the MTU of the interface, it fragments the packet and sends the fragments to the receiving system, which reassembles them into the original packet. If one of the fragments is lost in transmission, the incomplete packet is ultimately discarded by the receiver.

The maximum size of IP’s queue of packets received from the network interface is controlled by the ipqmaxlen parameter, which is set and displayed with *no*. If the number of packets in the input queue reaches this number, subsequent packets are dropped until the number goes down.

Send Flow

When the IP output routine receives a packet from UDP or TCP, it identifies the interface to which the mbuf chain should be sent, updates and checksums the IP part of the header, and passes the packet to the interface (IF) layer.

IP determines the proper device driver and adapter to use based on the network number. The driver interface table defines the maximum MTU for this network. If the datagram is less than the MTU size, IP adds the IP header in the existing mbuf, checksums the IP header, and calls the driver to send the frame.

If the datagram is larger than the MTU size (which only happens in UDP), IP fragments the datagram into MTU-sized fragments, appends an IP header (in an mbuf) to each, and calls the driver once for each fragment frame.

Receive Flow

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

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

4.5 LAN Adapters and Device Drivers

Many different kinds of LAN adapters are supported in the AIX environment. These adapters differ not only in the communications protocol and transmission medium they support but also in their interface to the I/0 bus and the processor. Similarly, the device drivers vary in the technique used to convey the data between memory and the adapter. The following high-level description applies to most adapters and device drivers, but details vary.

Send Flow

At the device-driver layer, the mbuf chain containing the packet is enqueued on the transmit queue. The maximum total number of output buffers that can be queued is controlled by the system parameter xmt_que_size. In some cases, the data is copied into driver-owned DMA buffers. The adapter is then signaled to start DMA operations.

At this point, control returns back up the path to the TCP or UDP output routine, which continues sending as long as it has more to send. When all data has been sent, control returns to the application, which then runs asynchronously while the adapter transmits data. When the adapter has completed transmission, it interrupts the system, and the device interrupt routines are called to adjust the transmit queues and free the mbufs that held the transmitted data.

Receive Flow

When frames are received by an adapter, they are transferred from the adapter into a driver-managed receive queue. The receive queue may consist of mbufs, or the device driver may manage a separate pool of buffers for the device; in either case, the data is in an mbuf chain when it is passed from the device driver to the IF layer.

Some drivers receive frames through DMA into a pinned area of memory and then allocate mbufs and copy the data into them. Drivers or adapters that receive large-MTU frames may have the frames DMA'd directly into cluster mbufs. The driver hands off the frame to the proper network protocol (IP in this example) by calling a demultiplexing function that identifies the packet type and puts the mbuf containing the buffer on the input queue for that network protocol. If no mbufs are available, or, if the higher-level input queue is full, the incoming frames are discarded.

Part 2. SP System Performance Tuning

Chapter 5. SP Performance Tuning Cycle

Tuning an SP system is an ongoing process that requires continuous review of the system settings and adjustments for changes in the computing environment. The following eight steps describe the *performance tuning cycle*:

1. Installation or migration

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

   - Commercial
   - Development
   - Scientific and Technical
   - Default

   Selecting a tuning file at this stage is not the end of all efforts, but the beginning. These files should be seen as base setup for getting a running system. You need to go through the rest of the SP performance tuning cycle and adjust the parameters according to your requirements. A sample tuning.cust file is located in the /usr/lpp/ssp/samples directory on the CWS.

2. Document your system

   When your system is alive and installed correctly, you should document your system settings and your network configuration. At this stage, the SP system is running, but you may not necessarily know its performance.

3. Evaluate the workload

   You need to evaluate system workload characteristics. You must know which type of environment you are running: is it a homogenous or consolidated type of environment?

4. Establish new tunable settings

   Based on the previous step, you may need to establish new values for the tunables.

5. Record and keep track of system settings

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

6. Apply and manage new settings

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

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

   For a small number of parameters that are not dynamically tunable, the values should be placed in the /etc/rc.net file. The following tunables are the only ones in this category:

   - arptab_nb
   - arptab_bsiz
   - arpqsize
   - ifsize

   Once any changes have been made to /etc/rc.net, the system must be rebooted before these variables take effect.

7. Monitor performance indices

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

Eventually, one of the following two situations may occur:

- Performance problems

  The system shows performance problems. Record and analyze the data that shows performance degradation. Perform AIX performance analysis, SP Switch analysis, SP log analysis, and so forth, and then go back to Step 3.

- System changes

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

Figure 16 illustrates this performance tuning cycle.

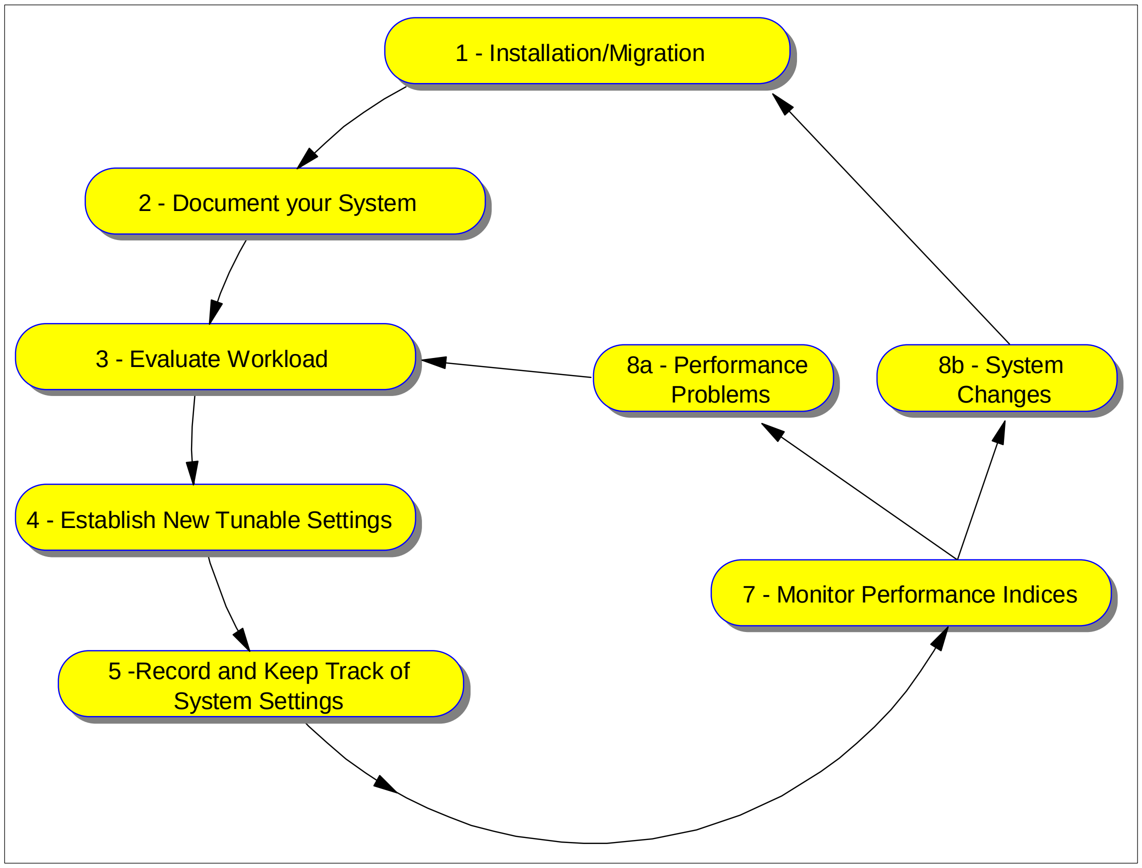

1 - Installation/Migration
2 - Document your System
3 - Evaluate Workload
4 - Establish New Tunable Settings
5 -Record and Keep Track of System Settings
7 - Monitor Performance Indices
8a - Performance Problems
8b - System Changes

Figure 16. Performance Tuning Cycle

Chapter 6. SP Network Tunables

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

This chapter discusses the way IBM RS/6000 SP systems are tuned for use within various environments. We first highlight general system recommendations and later describe specific tuning strategies.

6.1 Initial Considerations

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

6.1.1 General Tuning Recommendations

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

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

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

6.1.2 Consolidated System Challenges

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

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

6.1.3 System Topology Considerations

The most important consideration in configuring the SP Ethernet is the number of subnets configured (see Figure 18 on page 53). The routing through the Ethernet can be complicated because of the limitation on the number of simultaneous network installs per subnet. More information regarding the topology of the Ethernet can be found in *Planning Volume 1, Hardware and Physical Environment,* GA22-7280.

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

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

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

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

Several gateways or the SP router node should be configured if a large volume of traffic is expected. All traffic on these networks can be monitored using the standard network monitoring tools. Details about the monitoring tools and their usage can be found in *AIX Version 3.2 and 4.1 Performance Tuning Guide,* SC23-2365 or in Chapter 11, “IBM Performance Tools” on page 153.

6.2 AIX Network Tunables

The tunable values should be set to customized values during the installation process. See 6.4, “Tuning the SP Network for Specific Workloads” on page 59 for recommended default settings. Use the `no` command to display the current settings.

The most important tunables needed for the SP are:

thewall

Purpose:	This specifies the maximum amount of memory, in KB, that is allocated to the memory pool. In AIX Version 4.2.1 and earlier, the default value is 1/4 of real memory or 65536 (64 MB), whichever is smaller. In AIX Version 4.3.0, the default value is 1/4 of real memory or 131072 (128 MB), whichever is smaller. In AIX Version 4.3.1, the default value is 1/2 of real memory or 131072 (128 MB), whichever is smaller. In AIX Version 4.3.2 and later, the default value is 1/2 of real memory or 1048576 (1 GB), whichever is smaller. thewall is a runtime attribute.
Values:	Defaults: see *Purpose*.
Display:	`no -o thewall`
Change:	`no -o thewall=newvalue`
	Changes take effect immediately and are effective until the next reboot.
Tuning:	Increase size, preferably, to multiples of 4 KB.

sb_max

Purpose:	This provides an absolute upper bound on the size of TCP and UDP socket buffers per socket. Limits udp_sendspace, udp_recvspace, tcp_sendspace and tcp_recvspace. The units are in bytes.
Values:	Default: 65536
Display:	`no -o sb_max`
Change:	`no -o sb_max=newvalue`
	Changes take effect immediately for new connections. Change is effective until next reboot.
Tuning:	Increase size, preferably to multiples of 4096. Should be at least twice the size of the largest value for tcp_sendspace, tcp_recvspace, udp_sendspace, and udp_recvspace.

subnetsarelocal

Purpose:	This specifies that all subnets that match the subnet mask are to be considered local for purposes of using MTU instead of the maximum segment size (MSS).
Values:	Default: 1 (yes), Range: 0 or 1
Display:	`no -o subnetsarelocal`
Change:	`no -o subnetsarelocal=newvalue`
	Changes take effect immediately for new connections. Change is effective until next reboot.
Tuning:	This is a configuration decision with performance consequences. If the subnets have the same MTU, and subnetsarelocal is 0, TCP sessions may use an unnecessarily small MSS.

ipforwarding

Purpose:	This specifies whether the kernel should forward IP packets.
Values:	Default: 0 (no), Range: 0 or 1
Display:	`no -o ipforwarding`
Change:	`no -o ipforwarding=newvalue`
	Changes take effect immediately. Change is effective until next reboot.

Comment:	This is a configuration decision with performance consequences. Set to **1** for gateway nodes.

tcp_sendspace

Purpose:	Provides the default value of the size of the TCP socket send buffer in bytes.
Values:	Default: 16384, Range: 0 to 64 KB if rfc1323=0,
	Range: 0 to 4 GB if rfc2323=1.
Display:	`no -o tcp_sendspace`
Change:	`no -o tcp_sendspace=newvalue`
	Changes take effect immediately for new connections. Change is effective until next reboot.
Tuning:	Increase size, preferably to a multiple of 4096.

tcp_recvspace

Purpose:	This provides the default value of the size of the TCP socket receive buffer. The value is in bytes.
Values:	Default: 16384, Range: 0 to 64 KB if rfc1323=0,
	Range: 0 to 4 GB if rfc2323=1, should be less than sb_max.
Display:	`no -o tcp_recvspace`
Change:	`no -o tcp_recvspace=newvalue`
	Changes take effect immediately for new connections. Change is effective until next reboot.
Tuning:	Increase size, preferably to a multiple of 4096.

udp_sendspace

Purpose:	This provides the default value of the size of the UDP socket send buffer, in bytes.
Values:	Default: 9216, Range: 0 to 65536
Display:	`no -o udp_sendspace`
Change:	`no -o udp_sendspace=newvalue`
	Changes take effect immediately for new connections. Change is effective until next reboot.
Tuning:	Increase size, preferably, to a multiple of 4096. This should always be less than udp_recvspace but never greater than 65536.

udp_recvspace

Purpose:	Provides the default value of the size of the UDP socket receive buffer.
Values:	Default: 41600
Display:	`no -o udp_recvspace`
Change:	`no -o udp_recvspace=newvalue`
	Changes take effect immediately for new connections. Change is effective until next reboot.
Tuning:	Increase size, preferably to a multiple of 4096. Should always be greater than udp_sendspace and sized to handle as many simultaneous UDP packets as can be expected per UDP socket.

rfc1323

Purpose:	Value of 1 indicates that tcp_sendspace and tcp_recvspace sizes can exceed 64 KB.
	If the value is 0, the effective tcp_sendspace and tcp_recvspace sizes are limited to a maximum of 65535.
Values:	Default: 0, Range: 0 or 1
Display:	`no -o rfc1323`
Change:	`no -o rfc1323=newvalue`
	Changes take effect immediately for new connections. Change is effective until next reboot.
Comment:	Always set to 1.

tcp_mssdflt

Purpose:	This is the default maximum segment size used in communicating with remote networks.
Values:	Default: 512, Range: 512 to unlimited
Display:	`no -o tcp_mssdflt`
Change:	`no -o tcp_mssdflt=newvalue`
	Changes take effect immediately for new connections. Change is effective until next reboot.
Tuning:	Increase size, if practical. If set higher than MTU of the adapter, IP or intermediate router fragments packets.

spoolsize

Purpose:	This is the size of the SP Switch device driver send pool in bytes.
Values:	Default: 512 KB, Range: 512KB to 16 MB
Display:	`lsattr -E -l css0`
Change:	`chgcss0 -l css0 -a spoolsize=newvalue` Changes update the ODM; so, these changes will be permanent. A reboot of the system is necessary in order to apply any changes.

rpoolsize

Purpose:	This is the size of the SP Switch device driver receive pool in bytes.
Values:	Default: 512 KB, Range: 512KB to 16 MB
Display:	`lsattr -E -l css0`
Change:	`chgcss0 -l css0 -a rpoolsize=newvalue` Changes update the ODM; so, these changes will be permanent. A reboot of the system is necessary in order to apply any changes.

MTU

Purpose:	This limits the size of the packets that are transmitted on the network in bytes.
Values:	Default: adapter dependent, Range: 512 bytes to 65536 bytes
Display:	`lsattr -E -l interface` (for example: tr0)
Change:	`chdev -l interface -a mtu=newvalue` Because all the systems on the LAN must have the same MTU, they must change simultaneously. Change is effective across boots.
Tuning:	The default size should be kept.

6.2.1 TCP Maximum Segment Size (MSS)

The TCP protocol includes a mechanism for both ends of a connection to negotiate the maximum segment size (MSS) to be used over the connection. Each end uses the OPTIONS field in the TCP header to advertise a proposed MSS. The MSS that is chosen is the smaller of the values provided by the two ends.

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

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

6.2.2 Subnetting and the subnetsarelocal

Several physical networks can be made to share the same network number by subnetting. The easiest way to understand subnet addressing is to imagine that a site has a single class B IP network assigned to it, but it has two or more physical networks. Only local routers know that there are multiple physical nets and how to route among them.

Conceptually, adding subnets only changes the interpretation of the IP address slightly. Instead of dividing the 32-bit IP address into a network prefix and a host suffix, subnetting divides the address into a network portion and a local portion. The interpretation of the network portion remains the same as for networks that do not use subnetting. The interpretation of the local portion is left up to the site.

The example in Figure 17 shows subnet addressing with a class B address that has a 2-octet internet portion and 2-octet local portion. In the example, one octet of the local portion identifies a physical network, and the other octet identifies a host on that network.

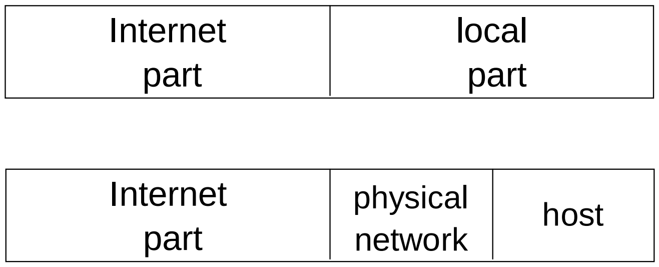


Figure 17. Subnet Addressing

In AIX, the no option subnetsarelocal specifies, on a system-wide basis, whether subnets are to be considered local or remote networks. With subnetsarelocal=1 (the default), Host A on subnet 1 considers Host B on subnet 2 to be on the same physical network.

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

This approach has several advantages:

- It does not require any static bindings; MSS is automatically negotiated.
- In order for small differences between adjacent subnets in the MTU to be handled appropriately, it does not disable or override the TCP MSS negotiation.

The disadvantages are:

- Potential IP router fragmentation when two high-MTU networks are linked through a lower-MTU network. Figure 18 illustrates this problem.

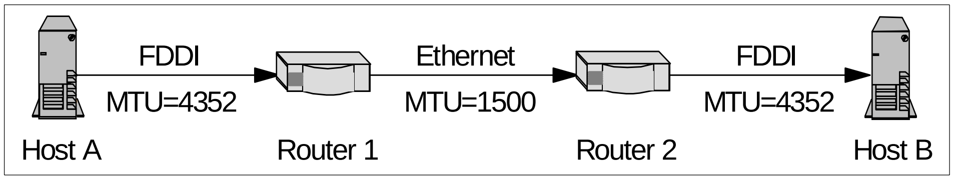


Figure 18. Inter-Subnet Fragmentation

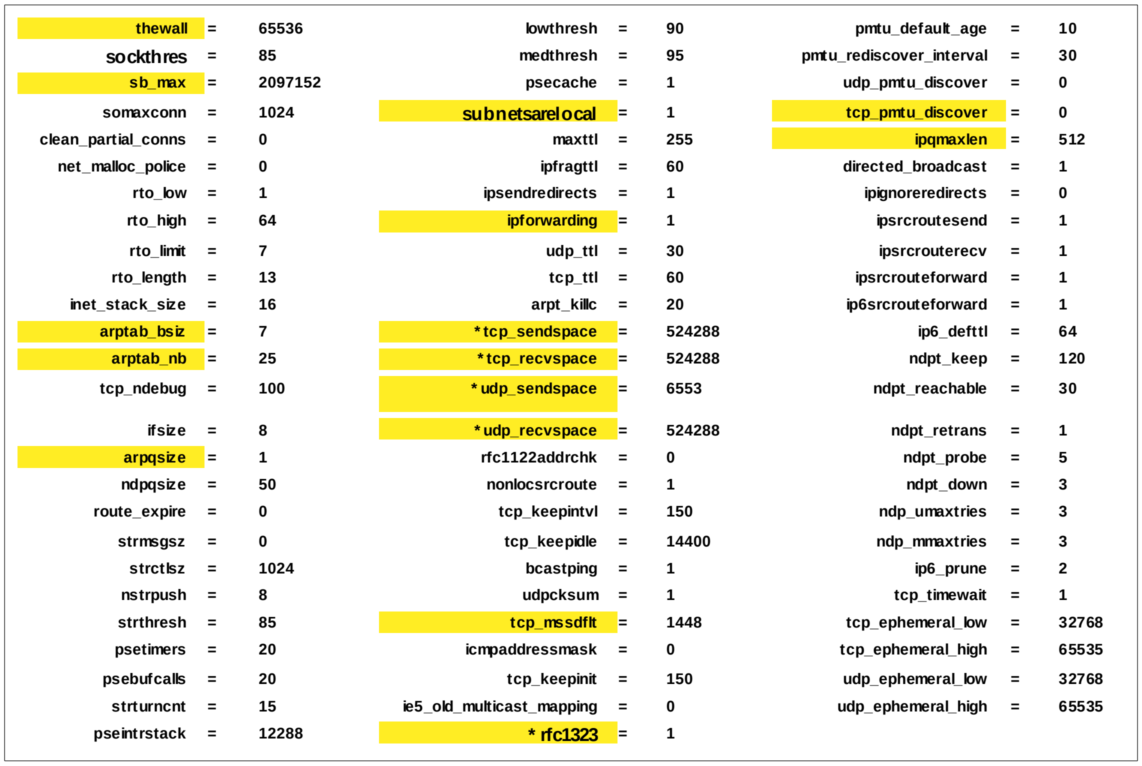
In this scenario, Hosts A and B establish a connection based on a common MTU of 4352. A packet going from A to B will be fragmented by

Router 1 and defragmented by Router 2, and the reverse will occur going from B to A.

• Source and destination must both consider subnets to be local.

6.3 SP System-Specific Tuning Recommendations

Let us have a closer look at the tunables that need some special observation in an SP environment. In the output of the `no -a` command taken from AIX 4.3.2 in Figure 19, the appropriate tunables are highlighted.



thewall	=	65536	lowthresh	=	90	pmtu_default_age	=	10
sockthres	=	85	medthresh	=	95	pmtu_rediscover_interval	=	30
sb_max	=	2097152	psecache	=	1	udp_pmtu_discover	=	0
somaxconn	=	1024	subnetsarelocal	=	1	tcp_pmtu_discover	=	0
clean_partial_conns	=	0	maxttl	=	255	ipqmaxlen	=	512
net_malloc_police	=	0	ipfragttl	=	60	directed_broadcast	=	1
rto_low	=	1	ipsendredirects	=	1	ipignoreredirects	=	0
rto_high	=	64	ipforwarding	=	1	ipsrcroutesend	=	1
rto_limit	=	7	udp_ttl	=	30	ipsrcrouterecv	=	1
rto_length	=	13	tcp_ttl	=	60	ipsrcrouteforward	=	1
inet_stack_size	=	16	arpt_killc	=	20	ip6srcrouteforward	=	1
arptab_bsiz	=	7	* tcp_sendspace	=	524288	ip6_defttl	=	64
arptab_nb	=	25	* tcp_recvspace	=	524288	ndpt_keep	=	120
tcp_ndebug	=	100	* udp_sendspace	=	6553	ndpt_reachable	=	30
ifsize	=	8	* udp_recvspace	=	524288	ndpt_retrans	=	1
arpqsize	=	1	rfc1122addrchk	=	0	ndpt_probe	=	5
ndpqsize	=	50	nonlocsrcroute	=	1	ndpt_down	=	3
route_expire	=	0	tcp_keepintvl	=	150	ndp_umaxtries	=	3
strmsgsz	=	0	tcp_keepidle	=	14400	ndp_mmaxtries	=	3
strctlsz	=	1024	bcastping	=	1	ip6_prune	=	2
nstrpush	=	8	udpcksum	=	1	tcp_timewait	=	1
strthresh	=	85	tcp_mssdflt	=	1448	tcp_ephemeral_low	=	32768
psetimers	=	20	icmpaddressmask	=	0	tcp_ephemeral_high	=	65535
psebufcalls	=	20	tcp_keepinit	=	150	udp_ephemeral_low	=	32768
strturncnt	=	15	ie5_old_multicast_mapping	=	0	udp_ephemeral_high	=	65535
pseintrstack	=	12288	* rfc1323	=	1			

Figure 19. Displaying Network Options

The following are specific details for setting the network tunables for the SP system:

thewall

This is set to at least 25 percent of real memory. The maximum value you can set this to is 65536 or 64 MB of memory except in AIX 4.3.2 where it is 131072 or 128 MB.

sb_max

This value should be at least twice the size of the largest value for tcp_sendspace, tcp_recvspace, or udp_recvspace. This ensures that if the buffer utilization is better than 50 percent efficient, the entire size of the tcp and udp byte limits can be utilized.

tcp_sendspace

This is never higher than the major network adapter transmit queue limit. To calculate this limit, use:

(major adapter queue size) * (major network adapter MTU)

tcp_recvspace

This is never higher than the major network adapter transmit queue limit.

(major adapter queue size) * (major network adapter MTU)

tcp_recvspace and tcp_sendspace

These tunables establish the size of the TCP window on a per-datagram socket connection. The effective size used is the least common denominator between the sending side tcp_sendspace and the receiving side tcp_recvspace. The size depends on the network you are sending over.

To properly set these tunables, you need a good understanding of the type of traffic your application will be sending. For peak switch performance, these need to be set to 512 KB or greater.

udp_sendspace

This is set to 65536 because anything beyond 65536 is essentially ineffective. Since UDP transmits a packet as soon as it gets any data, and since IP has an upper limit of 65536 bytes per packet, anything beyond 65536 runs the small risk of getting thrown away by IP.

udp_recvspace

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

rfc1323

If you are setting tcp_recvspace and tcp_sendspace to greater than 65536, you need to set rfc1323=1 on each side of the connection. Without having rfc1323 set on both sides, the effective values for tcp_recvspace and tcp_sendspace will be 65536.

tcp_mssdflt

This tunable is used to set the maximum packet size for communication with remote networks; however, only one value can be set even if there are several adapters with different MTU sizes. It is the same as the MTU for communication across a local network except for one small difference: The tcp_mssdflt size is for the size of only the data in a packet. You need to reduce the tcp_mssdflt for the size of any headers so that you send full packets instead of a full packet and a fragment. The way to calculate this is as follows:

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

which is:

MTU - 20 - 20 - 12, or MTU - 52

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

CSS MTU size

The MTU of a switch is 65520. We suggest that you keep this setting. However, under some circumstances (for example, when you want to avoid the Nagle algorithm causing very slow traffic), it may be necessary to reduce this value. You can reduce the MTU of a switch to 32678 with only a two to 10 percent loss in throughput. However, CPU utilization will be slightly higher due to the per-packet overhead.

To reduce the MTU of the switch, use:

```
ifconfig css0 mtu <new size>
```

This takes effect immediately and needs to be run as root user. You should always use the same MTU across all nodes in an SP.

6.3.1 Managing Tunable SP Parameters

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

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

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

- arptab_nb
- arptab_bsize
- arpqsize
- ifsize

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

- If you mess up rc.net, you can render the node unusable requiring a reinstall of the node.
- If you partially mess up rc.net, getting to the node through the console connection from the CWS can take several minutes or even over an hour. This is because parts of the initialization of the node try to access remote nodes or the CWS, and because rc.net is defective, each attempt to get remote data takes nine minutes to time out and fail.
- If you mess up tuning.cust, at least the console connection will work enabling you to log in through the CWS and fix the bad tunable settings.
- If you decide to create your own inittab entry to call a file with the tuning settings, future releases of PSSP will require a tuning.cust set of tunables to be run overriding your local modifications.

- Tuning.cust is run from rc.sp; so, it will always be run on a reboot.
- Tuning.cust includes a stop and start of inetd as of PSSP Release 2.4, which is required for all daemons to inherit the SP-specific tuning settings.

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

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

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

6.3.2 Initial Settings of SP Tunables

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

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

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

6.4 Tuning the SP Network for Specific Workloads

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

6.4.1 Tuning for Development Environments

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

Note

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

Table 3 on page 60 provides network-tunable settings designed as initial values for a development environment. These settings are only initial suggestions. Start with them and realize you may need to change them. Remember to keep track of your changes and document them.

Table 3. Software Development Tuning Parameters.

Parameter	Value
thewall	16384
sb_max	131072
subnetsarelocal	1
ipforwarding	1
tcp_sendspace	65536
tcp_recvspace	65536
udp_sendspace	32768
udp_recvspace	65536
rfc1323	1
tcp_mssdflt	1448
tcp_mtu_discover (new in AIX 4.2.1)	1
udp_mtu_discover (new in AIX 4.2.1)	1

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

6.4.2 Tuning for Scientific and Technical Environments

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

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

Experimentation shows that having between four and eight buffers allows TCP/IP to reach high transfer rates. The faster nodes require a greater number of buffers. However, if you set tcp_sendspace and tcp_recvspace to 655360 bytes, it can hurt the performance of the other network adapters connected to the node. This can cause adapter queue overflows described in "Adapter Queue Size" on page 74.

The following settings are only initial suggestions. Start with them and realize you may need to change them. Remember to keep track of your changes and document them.

Table 4. Scientific and Technical Environment Tuning Parameters

Parameter	Value
thewall	16384
sb_max	1310720
subnetsarelocal	1
ipforwarding	1
tcp_sendspace	655360
tcp_recvspace	655360
upd_sendspace	65536
upd_recvspace	655360
rfc1323	1
tcp_mssdflt	Varies depending on other network types
tcp_mtu_discover (new in AIX 4.2.1)	1
udp_mtu_discover (new in AIX 4.2.1)	1

6.4.3 Tuning for Commercial and Database Environments

The following table provides network tunable settings designed as initial values for commercial and database user environments. These initial settings are derived from the fact that commercial and database type applications generally have lots of network connections between nodes. For environments with lots of active connections, the tcp_sendspace and tcp_recvspace need to be adjusted so that the aggregate amount of the TCP window across all connections does not exceed the available buffer space for the SP Switch. In commercial and database environments where only a few connections are active, you can increase the tcp_sendspace and tcp_recvspace sizes to get better per-connection performance over the switch.

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

Table 5. Commercial and Database Environment Tuning Parameters.

Parameter	Value
thewall	16384
sb_max	1310720
subnetsarelocal	1
ipforwarding	1
tcp_sendspace	262144
tcp_recvspace	262144
udp_sendspace	65536
udp_recvspace	655360
rfc1323	1
tcp_mssdflt	1448
tcp_mtu_discover (new in AIX 4.2.1)	1
udp_mtu_discover(new in AIX 4.2.1)	1

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

If other applications with vastly different network characteristics are run on the same node, such as ADSM, or data mining type applications that tend to use few sockets, these settings may not provide peak performance. In these cases, the TCP window settings may have to be increased. Conflicts with the settings needed by ADSM can be resolved by having ADSM do its own socket level tuning. See Chapter 10, “ADSTAR Distributed Storage Manager (ADSM) Tuning” on page 147 for more information.

6.4.4 Tuning for Server Environments

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

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

The following table provides tunable network settings designed as initial values for server environments.

Table 6. Server Tuning Parameters.

Parameter	Value
thewall	65536
sb_max	1310720
subnetsarelocal	1
ipforwarding	1
tcp_sendspace	65536
tcp_recvspace	65536
udp_sendspace	65536
udp_recvspace	655360
rfc1323	1
tcp_mssdflt	1448
tcp_mtu_discover (new in AIX 4.2.1)	1
udp_mtu_discover (new in AIX 4.2.1)	1

6.4.5 Summary of Workload Tunables

Table 7 gives a combined overview of our tunables for the different environments.

Table 7. Summary of Workload Tunables.

Parameter	Commercial	Server	S&T	Dvlpmnt.
thewall	16384	65536	16384	16384
sb_max	1310720	1310720	1310720	131072
subnetsarelocal	1	1	1	1
ipforwarding	1	1	1	1
tcp_sendspace	262144	65536	655360	65536
tcp_recvspace	262144	65536	655360	65536
udp_sendspace	65536	65536	65536	32768
udp_recvspace	655360	655360	655360	65536
rfc1323	1	1	1	1
tcp_mssdflt	1448	1448	Varies depending on other network types	1448
tcp_mtu_discover (new in AIX 4.2.1)	1	1	1	1
udp_mtu_discover (new in AIX 4.2.1)	1	1	1	1

Chapter 7. Adapter Tuning

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

7.1 Maximum Transmission Unit (MTU)

The Maximum Transmission Unit (MTU) specifies the maximum size of packets (including all the protocol headers) that can be transmitted on a network. For an overview, see Figure 20 on page 69. All nodes and/or systems on the same physical network must have the same MTU. The MTU can be displayed using the `netstat -i` command. Table 8 gives an overview of common network adapters and their related MTU sizes.

Table 8. Maximum Transmission Units

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

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

7.2 Maximum Segment Size (MSS)

The Maximum Segment Size (MSS) is the largest segment or *chunk* of data that TCP will send to a destination. Figure 20 on page 69 shows the relations between MSS and MTU. The value of MSS is determined as follows:

1. If the destination is local, that is, if the network ID and the subnet ID of the destination IP address are the same as the local ones, the MSS value is calculated based on the MTU value of the outgoing interface as follows:

   MSS = MTU - (20 + 20) if rfc1323 = 0

   MSS = MTU - (20 + 20 + 12)if rfc1323 = 1

   since the TCP header is 20 bytes, and the IP header is also 20 bytes long. Furthermore, enabling rfc1323 costs an additional 12 bytes.
2. If the destination address is remote, that is, if the network ID of the destination IP address is different from the local one, TCP uses a global variable that determines the MSS.
3. If the destination has the same network ID as the local one but with a different subnet ID, the destination can be either local or remote. The *no* option of *subnetsarelocal* lets you specify whether subnets on the same network are local or remote. If local, follow item (1) above. If remote, follow item (2) above.

Since segmentation occurs at the TCP level, if the total packet is less than the MTU, IP does not do anything to the packet other than preappend a header and send the data to the interface layer. TCP on the receiving side will reassemble the packet in the correct sequence and deliver the data to the application.

7.3 TCP Data Flow

In order to comply with the MSS, TCP breaks the data into smaller pieces called segments. See also Figure 20 on page 69. The resulting IP datagram is 40 or 52 bytes larger: 20 bytes for the IP header and 20 bytes for the TCP header and an optional 12 bytes for rfc1323.

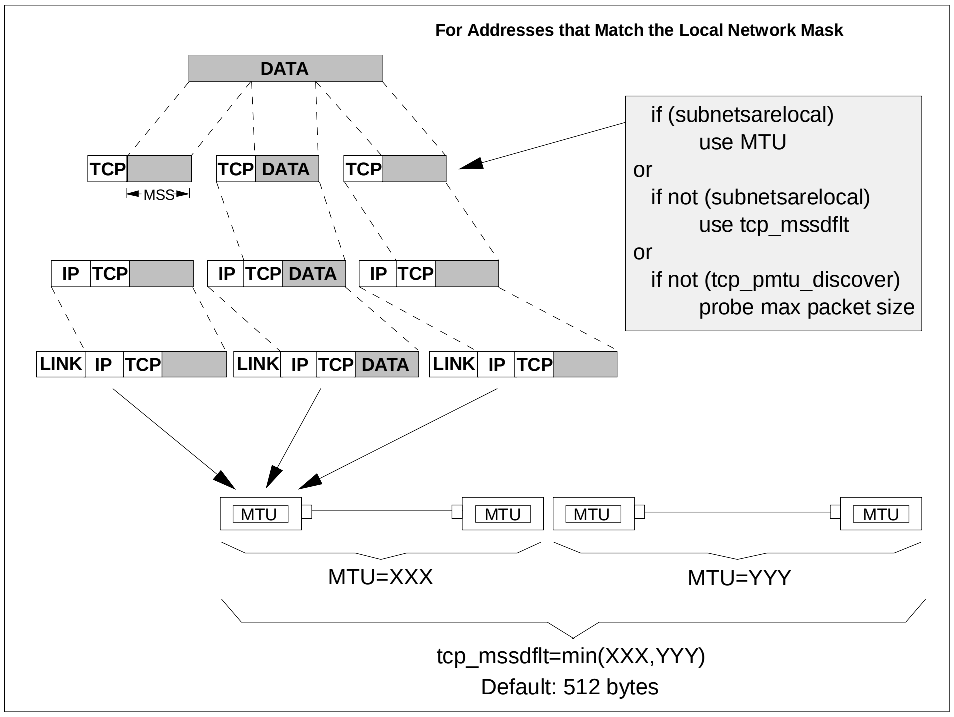


Figure 20. TCP Data Flow

When a connection is established, each end has the option of announcing the tcp_sendspace it is willing to receive depending upon its buffer space. Since the moving-window technique requires that the two systems be able to buffer the same amount of data, the effective window size is set to the lesser value in both directions. The nominally available extra space for buffering output shown in Figure 21 is never used.

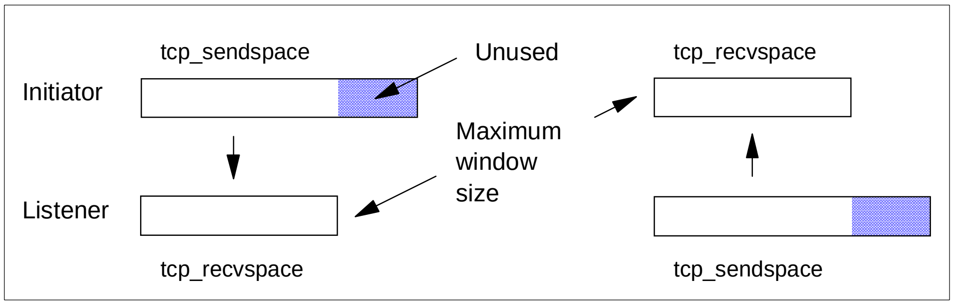


Figure 21. TCP Window Size

In general, the larger the MSS the better, as long as it is not so large that it causes fragmentation at the IP layer.

7.4 TCP Sliding Window

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

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

In the example in Figure 22, the sending application is sleeping because it has attempted to write data that would cause TCP to exceed the send socket buffer space (that is, tcp_sendspace). The sending TCP has sent the last part of rec5, all of rec6 and rec7, and the beginning of rec8. The receiving TCP has not yet received the last part of rec7 or any of rec8.

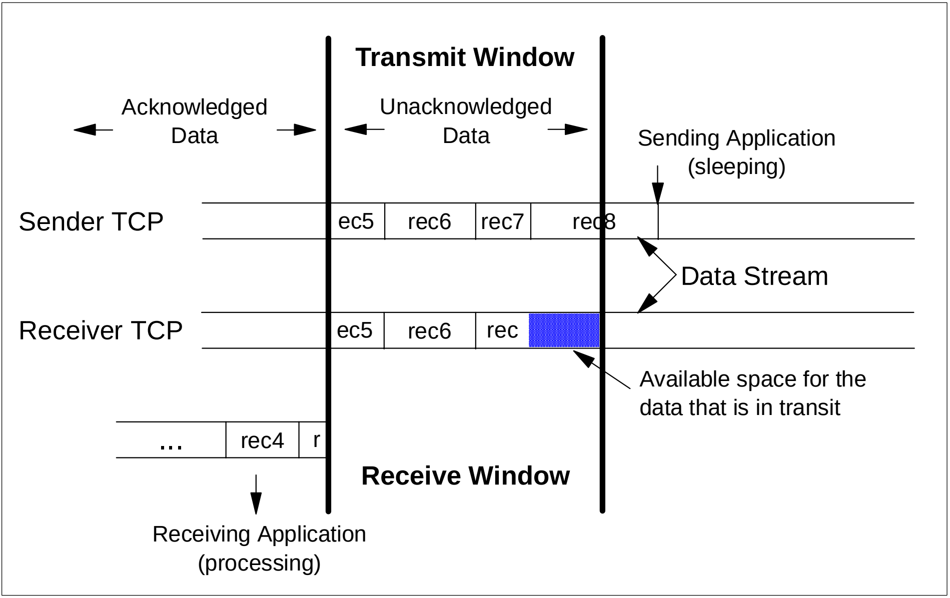


Figure 22. TCP Sliding Window

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

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

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

If the rfc1323 parameter is 1, the maximum TCP window size is 4 GB (instead of 64 KB). Figure 23 on page 72 illustrates this TCP enhancement.

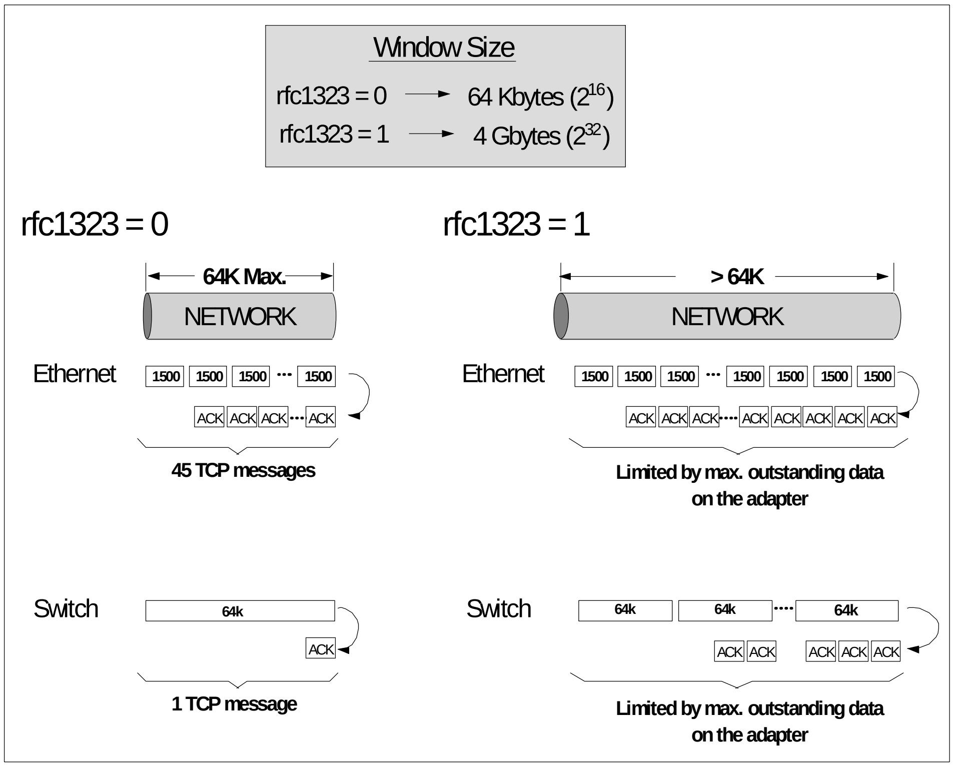

Window Size
rfc1323 = 0 → 64 Kbytes (2^16)
rfc1323 = 1 → 4 Gbytes (2^32)
rfc1323 = 0
rfc1323 = 1
64K Max.
> 64K
NETWORK
Ethernet
1500
ACK
45 TCP messages
Limited by max. outstanding data on the adapter
Switch
64k
1 TCP message

Figure 23. rfc1323 - TCP Extension

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

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

Also, many improvements have been made to the TCP sliding window mechanism. Here is a brief list of these improvements in RFC 1122:

- Silly Window Syndrome Avoidance Algorithm

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

- Delayed ACKs

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

- Nagle Algorithm

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

Note

Certain applications, such as X-Windows, do not work well with this mechanism. Thus, there must be an option to disable this feature. For this, you can use the TCP_NODELAY option of the setsockopt() call. For more information about the Nagle Algorithm, see “The Nagle Algorithm” on page 133.

- Congestion Avoidance

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

- Slow Start

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

7.5 Adapter Queue Size

The high throughput of a switch can cause problems with network adapters such as Ethernet, Token Ring, and FDDI connected to nodes acting as gateways on SP systems. There is a fixed number of adapter queue slots to stage packets in each network adapter device driver for traffic to that network. The transmit adapter queue length specifies the maximum number of packets for the adapter. The SP Switch send and receive pools are separate buffer pools as shown in Figure 24.

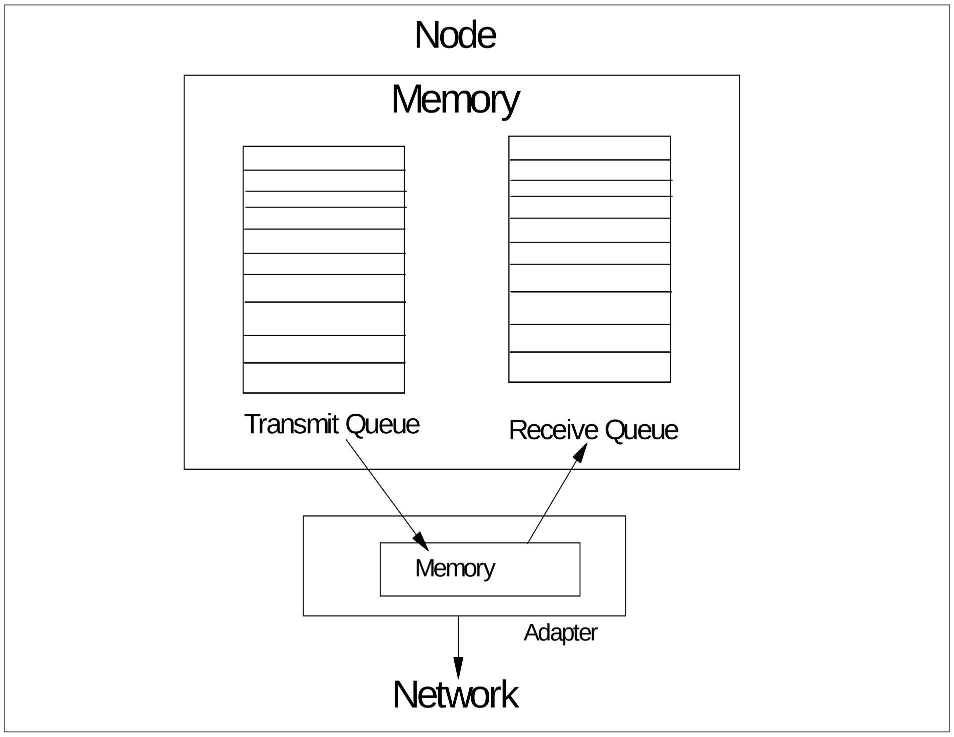


Figure 24. Adapter Queue Overview

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

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

Most communication drivers provide a set of tunable parameters to control transmit and receive resources. These parameters typically control the transmit queue and receive queue limits but may also control the number and size of buffers or other resources. They limit the number of buffers or packets that may be queued for transmit or limit the number of receive buffers that are available for receiving packets. For an example, see Table 9 on page 76. These parameters (for AIX 4.2.1 and newer) can be tuned to ensure enough queueing at the adapter level to handle the peak loads generated by the system or the network.

Table 9. Transmit Queue Size Examples

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

7.5.1 Transmit and Receive Queues

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

A typical example would be that you set your adapter queue length to 30. Assuming that the MTU of that adapter is 1500, you have set the maximum amount of data which that adapter can hold to 45,000 bytes. This is less than a single packet from a switch. Figure 25 on page 77 illustrates the different

MTU ratios. If you try and stage more packets to an adapter, the packets that arrive when this queue is full get thrown away.

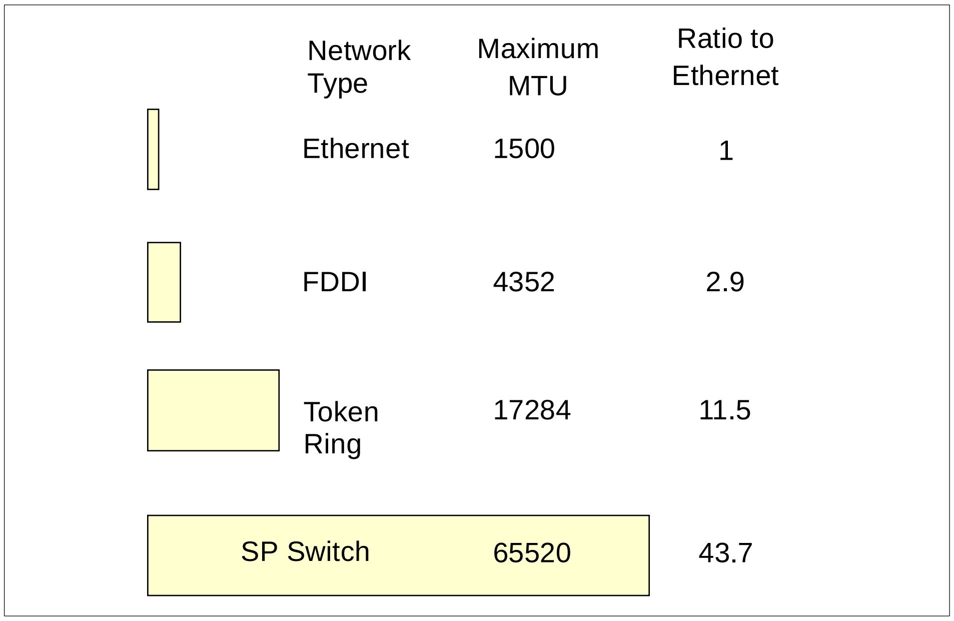


Figure 25. MTU Ratio

Receive Queues are the same as transmit hardware queues.

7.5.2 Displaying Adapter Queue Settings

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

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

and to display the current values, you can use

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

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

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

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

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

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

Figure 26. lsattr Command Output for an ATM Adapter

7.5.3 Changing Adapter Settings

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

For example, to change tx_que_size on en0 to 1024, use the following sequence of commands. Note that this driver only supports four different sizes; so, it is better to use SMIT to see the valid values.

```
# ifconfig en0 detach

# chdev -1 ent0 -a tx_que_size=1024

# ifconfig en0 up
```

7.5.4 Adapter Tuning Recommendations

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

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

7.6 Switch Adapter Tuning

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

7.6.1 Switch Adapter Pools

PSSP provides two variables to tune the sizes of the switch adapter pools:

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

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

To see the current send pool and receive pool buffer sizes as shown in Figure 27, issue the `lsattr -E -l css0` command.

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

Figure 27. Viewing Send and Receive Pool Buffer Sizes

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

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

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

The switch adapter pools reside in pinned kernel memory.

7.6.2 Switch Pool Allocation

The size of the buffers allocated by the switch device driver starts at 4096 bytes and increases to 65536 bytes in values of powers of 2. See Figure 28 on page 81 for an overview.

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

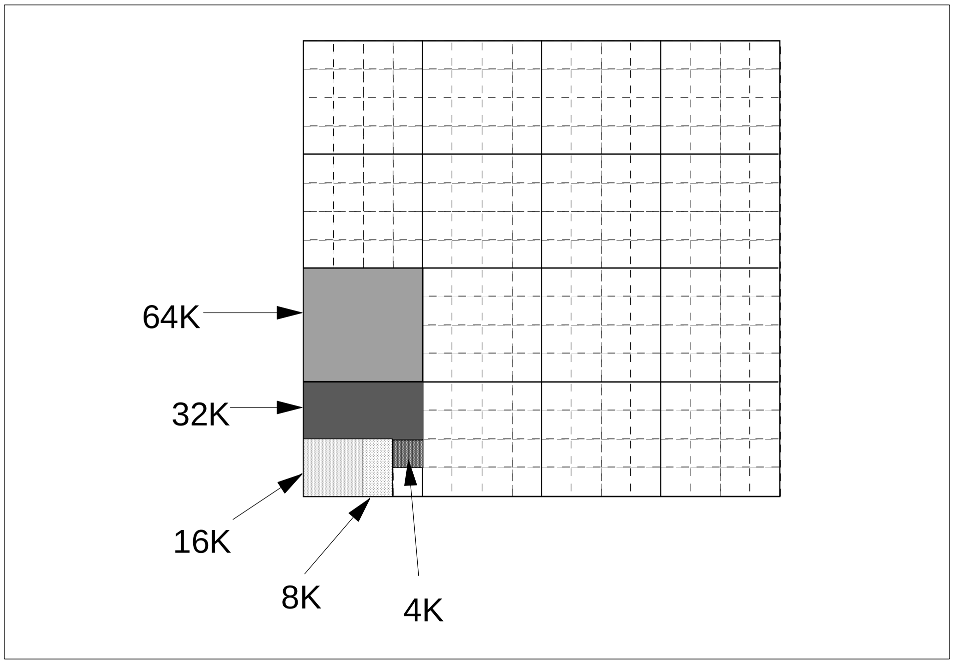


Figure 28. Switch Pool Allocation

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

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

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

7.6.3 Switch Buffer Pool Allocation Considerations

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

If there are a small number of active sockets, there is usually enough rpool and spool space that can be allocated. In systems where a node has a large number of sockets opened across the switch, it is very easy to run out of spool space when all sockets transmit at once. For example, 300 sockets, each sending 33 KB of data, will far exceed the 16 MB limit for the spool. Or, 1100 sockets, each sending 1 KB packets, will also exceed the maximum limit.

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

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

7.6.4 Sizing Send and Receive Pool Requirements

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

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

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

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

Use the vdidl3xx commands shown in Table 10 to check for IP pool size problems indicated by slow network traffic or ENOBUF errors.

Table 10. vdidl3xx Commands

SP Switch Adapter Type	Command
TB3	`vdidl3 -i`
SPSMX	`vdidl3mx -i`
TB3PCI	`vdidl3pci -i`

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

Figure 29 on page 84 is a sample output of the first table from the `vdidl3` command.

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

Figure 29. Output of the vdidl3 Command

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

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

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

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

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

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

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

comb This is currently not used.

freed This is currently not used.

7.6.5 Sample Switch Send Pool Size Estimate

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

Buffer Size profile:

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

Aggregate tcp_sendspace equivalent in packets: 1024 packets

Total send pool space:

No space needed for small packets	0 MB
512 - 8 KB buffers for 5 KB packets	4 MB
256 - 32 KB buffers for 32 KB packets	8 MB
Total send pool space needed	12 MB

If the number of packets staged at any one time is 1024, the spool initial setting should be 12582912 or 12 MB. None of the small packets needs any spool space; the 5 KB packets each use 8 KB out of the spool and need about 4 MB of space, and the 32 KB packets need about 8 MB of spool space. The total estimated pool size needed is 12 MB.

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

7.6.6 Reducing Send/Receive Pool Requirements

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

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

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

7.7 SP Ethernet Tuning

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

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

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

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

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

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

1. Create a name server within the SP system complex so that you are not at risk for traffic problems to an external name server.

2. Do not use the name server, but create an /etc/hosts file of all addresses that the SP system and the applications need, and change the nodes to resolve their addresses locally.
3. Specify the name lookup hierarchy to first look locally in the /etc/hosts file, then the name server.

7.8 Token-Ring Performance Tuning Recommendations

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

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

The application block size should be in multiples of 4096 bytes.

7.9 FDDI Performance Tuning Recommendations

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

Unless the LAN has extensive traffic to outside networks, the default MTU of 4352 bytes should be retained.

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

7.10 ATM Performance Tuning Recommendations

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

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

7.11 HIPPI Performance Tuning Recommendations

The default MTU of 65536 bytes should not be changed.

For maximum throughput, an application using TCP should write 65536 bytes at a time where possible.

Set sb_max to a value greater than 2*655360.

TCP and UDP socket send and receive space defaults should be set to more than 64 KB.

7.12 Escon Interface Tuning

To achieve peak TCP/IP throughput over a switch and through Escon to MVS, you need to make sure that the maximum packet size possible is used over the Escon connection. Table 11 lists all the necessary parameters to tune for a maximum packet size of 4096 bytes.

Table 11. Escon Interface Tuning Parameters.

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

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

Chapter 8. Global File Systems Tuning

In this chapter, we look at the most commonly used global file systems in an SP, NFS and GPFS. Since GPFS requires VSD, we also look at this subsystem.

8.1 Network File System Tuning on the SP

Network File System (NFS) is a distributed file system implementation that provides remote transparent access to files and file systems. Originally implemented and introduced as a product of Sun Microsystems, it has been implemented by many other vendors. Many of these implementations are licensed implementations of the Sun Microsystems source. AIX NFS is one such implementation.

8.1.1 NFS Overview

NFS operates on a client/server basis; that is, files that are physically resident on a server disk or other permanent storage are accessed through NFS on a client machine. For an illustration, see Figure 30 on page 90. In this sense, NFS is a combination of a networking protocol, client and server daemons, and kernel extensions.

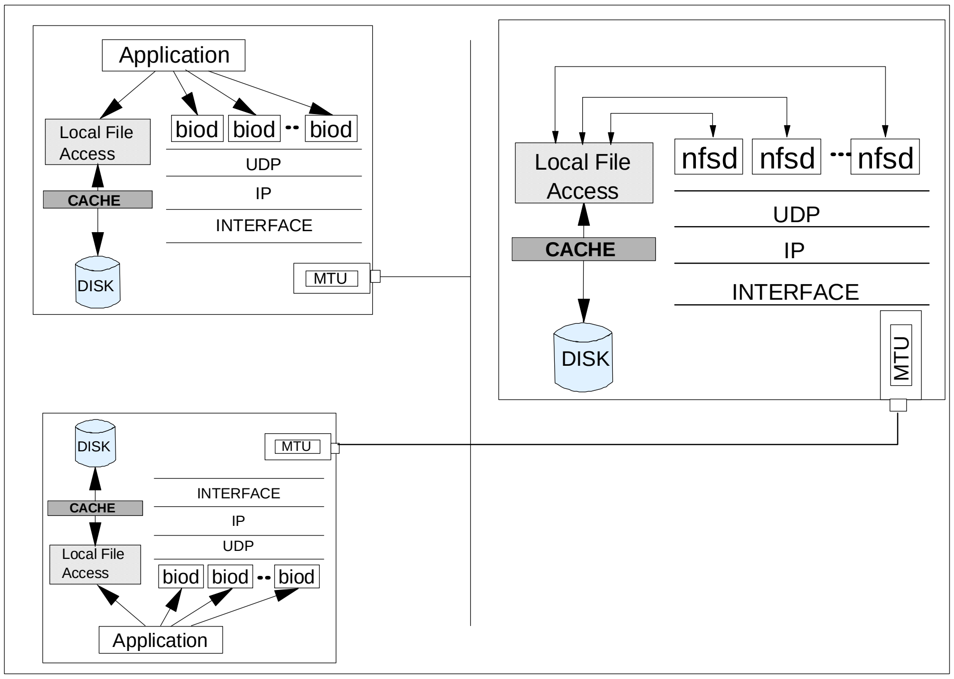


Figure 30. NFS Overview

8.1.1.1 The Role of nfsd and biod

nfsd is a server daemon and biod is a client daemon.

On the server side, the receipt of one NFS protocol request from a client requires an nfsd daemon until that request is satisfied and the results of the request processing are sent back to the client. The nfsd daemons are active agents providing NFS services.

On the client side, one biod (block I/O daemon) is required to send one read or write request to the server. Biods are only used for reading and writing files. Other NFS operations, such as name lookup, get file attributes, and so on are sent directly to the server from the AIX NFS client kernel extension.

8.1.2 Large-Scale Environment Considerations

When considering overall aspects of NFS performance, it is important to understand your system environment. Some of the tuning techniques that are discussed will help overall NFS performance. Other techniques may be unnecessary or may degrade performance when applied in the wrong environment.

The first and most important step in planning for a large-scale NFS installation is to make a thorough evaluation of how the application environment and user requirements translate into server requirements.

If there is an existing NFS environment that is running an application load similar to the one that is being planned, it is a simple process to get a profile of the workload using the `nfsstat -s` command on the server. If there is more than one server running with the expected application load, both should be monitored.

8.1.3 NFS Troubleshooting

NFS read and write requests generally involve the greatest latencies and move the largest volumes of data; so, if problems occur, they usually show up in reading or writing. Typically, performance starts to degrade when an NFS client begins getting *timeouts and retransmits*.

If something happens on the network that causes even one fragment of a read reply to be lost, the IP layer on the client will throw away the remaining fragments. In that case, the whole read request will have to be rerequested and retransmitted. The rerequest will be generated by the biod that made the initial request if it does not receive the reply in a fixed amount of time. But, when a timeout occurs, one of two actions can be taken: It can try the request again (a retransmit) or it can just give up and return an error.

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

It is important to understand that if a packet is lost on the network or dropped at the server, the full timeout interval will expire before the packet is retransmitted from the client. There is no intermediate ack mechanism that will inform the client, for example, that the server only received five of the expected six write fragment packets.

When you consider that you expect RPC requests to be normally completed in some very small fraction of a second, it is easy to see how repeated timeouts will have huge impacts on performance. Timeouts and the relatively large latencies involved when they occur are the basis for most poor performance. The challenge of diagnosing a large number of performance problems is determining exactly where the packets are being dropped and/or delayed.

Statistics can be queried at several different network and system layers to determine whether there are network failures, all of which can affect NFS.

8.1.4 Checklist for NFS Tuning

In this section, we develop a checklist for your NFS tuning work.

Use nfsstat output on the client to watch for timeout/retransmit occurrences. Timeouts and retransmits indicate poor performance and must be eliminated for optimum results. Also, use nfsstat output to correlate timeouts/retransmits on clients with server drop statistics gathered from the `netstat` command (socket buffer overflows, Oerrs) to make sure that the server statistics are indicating NFS problems.

Make sure the server is not just overloaded. Use standard performance evaluation methods and tools such as vmstat and iostat to see if the server is CPU or I/O bound.

Check for NFS UDP buffer overflows

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

Check for mbuf problems

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

8.1.5 Dropped Packets

Packets can be dropped at the client side, the server side, or somewhere on the network.

8.1.5.1 Packets Dropped by the Client

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

Packets are more commonly dropped either by the network or by the server.

8.1.5.2 Packets Dropped by the Server

There are two cases where servers will drop packets under heavy loads.

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

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

NFS packets will usually be dropped at the socket buffer only when a server has a lot of NFS write traffic. The NFS server uses a UDP socket attached to NFS port 2049, and all incoming data is buffered on that UDP port. The default size of this buffer is 60000 bytes. We can do some quick math: Dividing that number by the size of the default NFS write packet (8192), we find that it will take only eight simultaneous write packets to overflow that buffer. That could be done by just two NFS clients (with the default configurations).

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

Each of the two situations is handled differently. In the case of high volume, it may be sufficient to simply increase the number of nfsd daemons running on the system. Since there is no significant penalty for running with more nfsd daemons on an AIX machine, this should be tried first.

In the case of high burst traffic, the only solution is to make the socket bigger in the hope that some reasonable size will be sufficiently large enough to give the nfsd daemons time to catch up with the burst. Memory dedicated to this socket will not be available for any other use; so, it must be noted that a tuning objective of total elimination of socket buffer overflows by making the socket larger may result in this memory being underutilized the vast majority of the time. A cautious administrator will watch the socket buffer overflow statistic, correlate it with performance problems, and determine how large to make the socket buffer.

8.1.5.3 Dropped Packets On the Network

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

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

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

8.1.5.4 NFS Server Statistics

NFS gathers statistics on types of NFS operations performed, along with error information and performance indicators. The `nfsstat` command can be used to identify network problems and give an idea of the type of nfs operations taking place on your system.The `nfsstat` command displays this information categorizing it under the subtitles Server rpc, Server nfs, Client rpc, and Client nfs.

The NFS server displays the number of NFS calls received (calls) and rejected (badcalls) due to authentication, as well as the counts and percentages for the various kinds of calls made.

The example in Figure 31 on page 95 shows the server part of the `nfsstat` command.

```
# nfsstat -s
Server rpc:
Connection oriented:
calls      badcalls   nullrecv   badlen     xdrcall    dupchecks  dupreqs
0          0          0          0          0          0          0
Connectionless:
calls      badcalls   nullrecv   badlen     xdrcall    dupchecks  dupreqs
2490       0          0          0          0          0          0

Server nfs:
calls      badcalls   public_v2  public_v3
2490       0          0          0
Version 2: (0 calls)
null       getattr    setattr    root       lookup     readlink   read
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
wrcache    write      create     remove     rename     link       symlink
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
mkdir      rmdir      readdir    statfs
0 0%       0 0%       0 0%       0 0%
Version 3: (2490 calls)
null       getattr    setattr    lookup     access     readlink   read
0 0%       34 1%      0 0%       1 0%       1 0%       0 0%       2451 98%
write      create     mkdir      symlink    mknod      remove     rmdir
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
rename     link       readdir    readdir+   fsstat     fsinfo     pathconf
0 0%       0 0%       0 0%       0 0%       3 0%       0 0%       0 0%
commit
0 0%
```

Figure 31. Dropped Packets and Server Overruns

PC output for server (-s):

calls	This is the total number of RPC calls received from clients.
badcalls	This is the total number of calls rejected by the RPC layer.
nullrecv	This is the number of times an RPC call was not available when it was thought to be received.
badlen	This indicates packets truncated or damaged; It is the number of RPC calls with a length shorter than a minimum-sized RPC call.
xdrcall	This is the number of RPC calls whose header could not be External Data Representation (XDR) decoded.

dupchecks	This is the number of RPC calls that did a look-up in the duplicate request cache.
dupreqs	This is the number of duplicate RPC calls found.

It also displays a count of the various kinds of calls and their respective percentages.

NFS performance can be increased by adding additional nfsd daemons. But watch the nullrecv column in the nfsstat -s output. If the number starts to grow, it may mean there are too many nfsd daemons. However, this is usually not the case on AIX NFS servers as much as it could be the case on other platforms. The reason for this is that not all nfsd daemons are awakened at the same time when an NFS request comes into the server. Instead, the first nfsd wakes up, and, if there is more work to do, this daemon wakes up the second nfsd and so on.

8.1.5.5 NFS Client Statistics

Of particular interest in diagnosing performance problems is the RPC statistical data. The fields reporting *timeout* and *retrans* are particularly useful in gross monitoring of performance and evaluation of testing.

For an NFS client, if you see incidents of timeouts and retransmits, and the numbers are roughly equivalent, you can be assured that there are packets being dropped. Figure 32 shows the client part of the `nfsstat` command.

```
# nfsstat -c
Client rpc:
Connection oriented:
calls      badcalls   badxids    timeouts   newcreds   badverfs   timers
0          0          0          0          0          0          0
nomem      cantconn   interrupts
0          0          0
Connectionless:
calls      badcalls   retrans    badxids    timeouts   newcreds   badverfs
4420       0          1          0          1          0          0
timers     nomem      cantsend
5          0          0
Client nfs:
calls      badcalls   clgets     cltoomany
4398       0          0          0
Version 2: (7 calls)
null       getattr    setattr    root       lookup     readlink   read
0 0%       6 85%      0 0%       0 0%       0 0%       0 0%       0 0%
wrcache    write      create     remove     rename     link       symlink
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
mkdir      rmdir      readdir    statfs
0 0%       0 0%       0 0%       1 14%
Version 3: (4393 calls)
null       getattr    setattr    lookup     access     readlink   read
0 0%       106 2%     0 0%       18 0%      22 0%      0 0%       4194 95%
write      create     mkdir      symlink    mknod      remove     rmdir
0 0%       1 0%       0 0%       0 0%       0 0%       0 0%       0 0%
rename     link       readdir    readdir+   fsstat     fsinfo     pathconf
0 0%       0 0%       0 0%       5 0%       46 1%      1 0%       0 0%
commit
0 0%
```

Figure 32. Dropped Packets Overrun (Client Side)

The severity of the performance degradation resulting from the dropped packets is dependent upon the number of packets dropped and the time windows during which they were dropped. It is usually not wise to make judgments about performance without being able to directly correlate the two, and this generally requires a repeatable test. For this reason, the most valuable use of `nfsstat` is in performance tuning evaluation. The objective should be to generate as much throughput as possible while avoiding dropped packets.

RPC output for the client (-c):

calls	This is the total number of RPC calls made to NFS.
badcalls	This is the total number of calls rejected by the RPC layer.
retrans	This is the number of times a call had to be retransmitted due to a timeout while waiting for a reply from the server. This is applicable only to RPC over connectionless transports.

badxid	This is the number of times a reply from a server was received that did not correspond to any outstanding call. This means the server is taking too long to reply.
timeout	This is the number of times a call timed out while waiting for a reply from the server.
newcred	This is the number of times authentication information had to be refreshed.
badverfs	This is the number of times a call failed due to a bad verifier in the response.
timers	This is the number of times the calculated timeout value was greater than or equal to the minimum specified timeout value for a call.
cantconn	This is the number of times a call failed due to a failure to make a connection to the server.
nomem	This is the number of times a call failed due to a failure to allocate memory.
interrupts	This is the number of times a call was interrupted by a signal before completing.
cantsend	This is the number of times a send failed due to a failure to make a connection to the client.

It also displays a count of the various kinds of calls and their respective percentages.

For performance monitoring, nfsstat -c will give information on whether the network is dropping packets. A network may drop a packet if it cannot handle it. Dropped packets may be the result of the response time of the network hardware or software or an overloaded CPU on the server. Dropped packets are not actually lost since a replacement request is issued for them.

The retrans column in the RPC section displays the number of times requests were retransmitted due to a timeout while waiting for a response. This is related to dropped packets. If the retrans number consistently exceeds five percent of the total calls in column one, it indicates a problem with the server keeping up with demand. Use vmstat, netpmon, and iostat on the server machine to check the load.

8.1.6 Check for NFS UDP Socket Buffer Overflows

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

There are two things that can fix NFS socket buffer overruns. First, try just increasing the number of nfsd daemons that are being run on the server. If that does not cure the problem, you must adjust two kernel variables, *sb_max* (socket buffer max) and *nfs_socketsize* (the size of the NFS server socket buffer). Use the `no` command to increase sb_max. Its default value is 65536. Use the `nfso` command to increase the *nfs_socketsize* variable. Its default value is 60000.

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

You must restart the nfsd daemons after adjusting the sb_max and nfs_socketsize variables in order for them to take effect. Do stopsrc -s nfsd; startsrc -s-nfsd. If the nfsd daemons do not start, you have made a mistake in setting one of the two variables and sb_max is probably not greater than nfs_socketsize.

Configurations using the `no` and `nfso` command must be repeated every time the machine is booted. Put them in the rc.nfs file right before the nfsd daemons are started and after the biod daemons are started. *The position is important*.

8.1.7 Number of NFS Daemons

This is one of the most often asked questions regarding NFS tuning. Unfortunately, there is no simple answer.

On the server side, the receipt of any one NFS protocol request from a client requires the dedicated attention of an nfsd daemon until that request is satisfied and the results of the request processing are sent back to the client.

On the client side, one biod (block I/O daemon) is required to send any one read or write request to the server. Biods are only used for reading and writing files. Other NFS operations, such as name lookup, get file attributes, and so on, are sent directly to the server from the AIX NFS client kernel extension.

Careful consideration of the above two paragraphs may help make it clear why specifying a *correct* number of nfsd or biod daemons in any given environment is impossible without a thorough evaluation of the aggregate workload. To operate each biod, will tie up exactly one nfsd, but other nfsd daemons may be utilized by operations that were not originated by a biod.

Here is a summary of the discussed subjects:

- The default number of biod daemons is 6 (threads NFSv3).
- The default number of nfsd daemons is 8 (threads NFSv3).
- Each biod requires an nfsd on the server side.
- On the server, not all nfsd daemons will be attending a biod request.
- When sizing, consider server capabilities and client usage.

Transition Statement - If you have done all you can in increasing nfsd daemons or decreasing biod daemons and are still getting socket buffer overflows, you may need to increase the NFS socket buffer. The `nfso` command can be used to do this.

8.1.8 The nfso Command

The `nfso` command can be used to configure NFS attributes. It sets or displays network options in the currently running kernel. Therefore, the command must run after each system startup or network configuration.

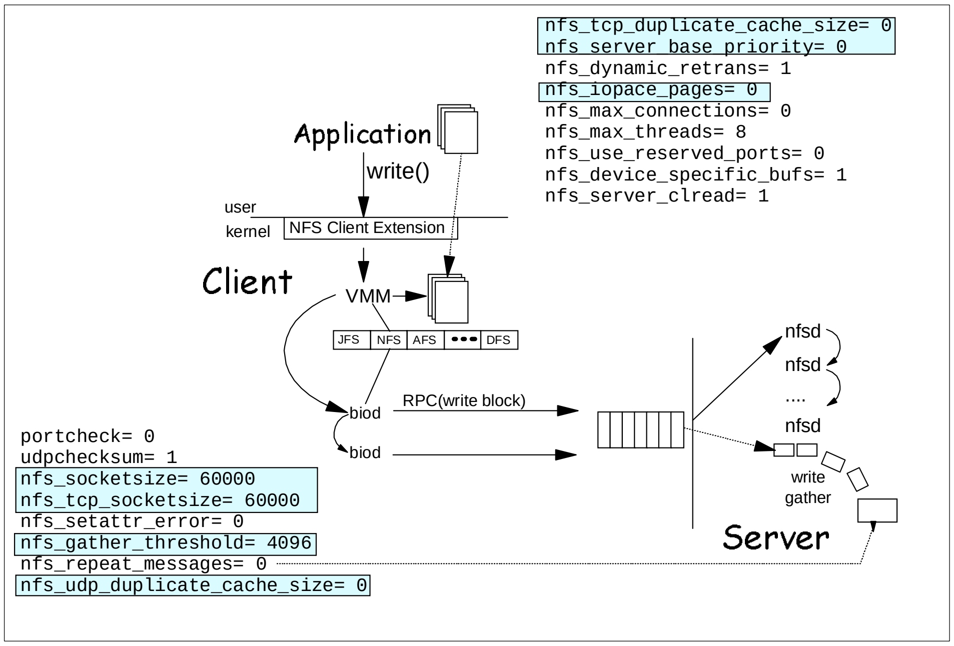


Figure 33. The nfso Command

Note

The `nfso` command performs no range checking. If used incorrectly, it can make your system inoperable.

The nfso parameters and their values can be displayed with nfso -a.

Most NFS attributes are runtime attributes that can be changed at any time with the exception of nfs_socketsize, which needs NFS to be stopped first and restarted afterwards.

To display or change a specific parameter, use the `nfso -o` command:

```
# nfso -o portcheck
portcheck= 0
# nfso -o portcheck=1
```

The parameters can be reset to their default values with the `-d` option:

```
# nfso -d portcheck
# nfso -o portcheck
portcheck= 0
```

nfso Command Parameters

portcheck

This checks whether an NFS request originated from a privileged port. The default value of 0 disables the port checking that is done by the NFS server. A value of 1 directs the NFS server to do port checking on the incoming NFS requests.

This is a configuration decision with minimal performance consequences.

udpchecksum

This performs the checksum of NFS UDP packets. The default value of 1 directs the NFS server or client to build UDP checksums for the packets that it sends to the NFS clients or servers. A value of 0 disables the checksum on UDP packets from the NFS server or client.

Turning this off is not recommended. Make sure this value is set to 1 in any network where packet corruption may occur. Slight performance gains can be realized by turning it off, but this increases the chance of data corruption.

nfs_socketsize

This sets the queue size of the NFS server UDP socket. This socket is used for receiving the NFS client requests and can be adjusted so that the NFS server is less likely to drop packets under heavy load. The value of the nfs socketsize variable must be less than the sb_max option, which can be manipulated by the no command.

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

nfs_setattr_error

When set to a value of 1, the NFS server ignores invalid setattr requests. This is provided for certain Personal Computer applications. The default value is 0.

Tuning this parameter should not increase the performance.

nfs_gather_threshold

Determines when to attempt to gather write requests to a file. AIX attempts to increase throughput to the disks by implementing *write gather* heuristics on the server. The fundamental idea is that if the server sees multiple 8192 write packets coming in, chances are that this is a sequential file write, and, rather than sync each 8 KB of data individually, it strives to coalesce these writes into larger blocks that can be more efficiently handled by the file system and disk subsystems. Up to six write requests will be gathered and synced together. When the sync returns, the nfsd daemons are all free to reply to the client. If the size of the NFS write request is less than the value of the nfs_gather threshold option, the NFS server writes the data and immediately responds to the NFS client. If the size of the NFS write request is equal to or greater than the value of this option, the NFS server writes the data and waits for a small amount of time before responding to the NFS client.

The write gathering can be a performance advantage for sequential writes but can produce slight performance decreases for random writes. Look at the following two scenarios:

1. Delays are observed in responding to RPC requests, particularly those where the client is exclusively doing nonsequential writes, or the files being written are written with file locks held on the client.
2. Clients are writing with write sizes smaller than 4096 KB, and write gather is not working.

If write gather is to be disabled, change the nfs_gather_threshold to a value greater than the largest possible write. For AIX V4 running NFS Version 2, that would be 8192. So, changing the value to 8193 disables write gather. Use this for the situation described above in the first scenario. If write gather is being bypassed due to a small write size, say 1024 KB (described in the second scenario), change the write gather parameter to gather smaller writes. For example, set it to 1024.

nfs_repeat_messages

Checks for duplicate NFS messages. This option is used to avoid displaying duplicate NFS messages. When set to a value of 1, all NFS messages are printed to the screen. If set to a value of 0, duplicate messages appearing one after the other are not printed to the screen. Only the first message of such a sequence is displayed. When a different message appears, a message similar to the following will be displayed:

```
LastNFS message repeated n times
```

Tuning this parameter does not increase performance.

nfs_duplicate_cache_size

The `nfso` command cannot be used to decrease the nfs_duplicate_cache_size value. Any attempt will fail; the system has to be rebooted. The duplicate cache size should be increased for very fast or busy servers that have a high throughput capability. The duplicate cache is used to allow the server to correctly respond to NFS client retransmissions. If the server flushes this cache before the client is able to retransmit, the server may respond incorrectly, and the client can observe anomalous NFS behavior. Therefore, if the server can process 1000 operations before a client retransmits, the duplicate cache size needs to be increased.

Calculate the number of NFS operations that are being received per second at the NFS server and multiply it by four. This will produce a duplicate cache size that should be sufficient to allow correct response from the NFS server. The operations that are affected by the duplicate cache are the following: *setattr*, *write*, *create*, *remove*, *rename*, *link*, *symlink*, *mkdir*, and *rmdir*.

nfs_server_base_priority

If this value is set, the nfsd processes use it as their base priority. Acceptable values are from 31 to 126. The default value is 0, which means that the nfsd processes have a regular floating priority. Therefore, as they increase their cumulative CPU time, their priority will change. This parameter can be used to set a static priority for the nfsd daemons. Other values within the acceptable range will be used to set the priority of the nfsd daemon when an NFS request is received at the server. This option can be used if the NFS server is overloading the system (lowering or making the nfsd daemon less favored). It can also be used if it is desired that the nfsd daemons be one of the most favored processes on the server. Care must be taken when setting the parameter because it may render the system almost unusable by other processes. This can occur if the NFS server is very busy and, essentially, locks out other processes from having runtime on the server.

nfs_dynamic_retrans

With this parameter set to 1, the NFS client attempts to adjust its timeout behavior based on past NFS server responses. It allows the NFS client to automatically decrease the size of NFS read/write packets to attempt to respond to network or server load problems. This also allows the NFS client the ability to vary the timeout value used for the retransmission. All of this is done based on a cumulative history of the NFS server's response time. In most cases, this parameter does not need to be adjusted. There are some instances where the straightforward timeout behavior is desired for the NFS client. In these cases, the value should be set to **0** before mounting file systems.

nfs_iopace_pages

This is the maximum number of dirty pages that the NFS client will flush to the NFS server at one time. This is often useful when, for instance, large compilations of images are flushed by the binder, and interactive performance suffers or when an application writes a large file to an NFS-mounted file system. That file data is written to the NFS server when the file is closed. In some cases, the resource it takes to write that file to the server may prevent other NFS file I/O from occurring. The default value for this parameter limits the number of 4 KB pages written to the server to 32. The NFS client will schedule 32 pages for writing to the server and then waits for these to complete before scheduling the next 32. The default value is usually sufficient for most environments. The value should be decreased if there are large amounts of contention for NFS client resources. If there is low contention, the value can be increased.

8.1.9 Mount Options That Affect Performance

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

Before you start adjusting mount options, you have to know what it is you are trying to achieve with respect to packet delivery and packet turnaround on the server or network. Most of the NFS-specific mount options will be utilized if your goal is to decrease the load on the NFS server, or to work around network problems.

The NFS performance-specific mount options are all specified as a list entry on the -o option for mount. Options you enter for the -o option on the command line should be separated only by a comma, not a comma and a space.

The most useful options are the ones for changing the read and write size values. Often, the rsize and wsize mount options are decreased in order to decrease the read/write packet that is sent to the server. There can be two reasons why you might want to do this.

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

On the other hand, if your NFS file system is mounted across a high-speed network, such as the SP Switch, then larger read and write packet sizes would enhance NFS file system performance. With NFS Version 3, rsize and wsize can be set as high as 64 KB. (With NFS Version 2, the largest that rsize and wsize can be is 8 KB.) This and other NFS Version 3 enhancements are discussed later in this unit.

8.1.10 Configuring Server Disk Usage

Since the writes at the server must be performed synchronously, writes are usually disk-bound. In addition, for each write at the server, there has to be a JFS log file update that must also be done synchronously. This results in two sequential, synchronous writes if the log logical volume is on the same disk as the writable file system. Large performance increases can often be realized by putting the log logical volume on a separate physical volume (disk) so that the subsystems can increase the parallelism of these two synchronous operations.

Often it is necessary to achieve parallelism on data access beyond that described in the above paragraph regarding the log file. Concurrent access to a single file system on a server by multiple clients or multiple client processes can result in throughput being bottlenecked on the disk I/O for a particular device. You can use the `iostat` command to evaluate disk loading.

For large NFS servers, the general strategy should be to flatten out the disk I/O demand across as many disk/disk adapter devices as possible. This results in greater parallelism and the ability to run greater numbers of nfsd daemons. On a system where disk I/O has been well distributed, it is possible to reach a point where CPU load becomes the limiting factor on the server's performance.

8.1.11 Network Locking Performance Implications

Regardless of the client and server performance capacity, all operations involving NFS file locking will probably seem unreasonably slow. There are several technical reasons for this, but they are all driven by the fact that, if a file is being locked, special considerations have to be taken to ensure that the file is synchronously handled on both the read and write sides. This means there can be no caching of any file data at the client. It also means that caching of file attributes is turned off. All file operations go to a fully synchronous mode with no caching whatsoever. You might suspect that an application is doing network file locking if it is operating over NFS and shows abnormally poor performance compared to other applications on the same client/server pair.

8.1.12 NFS Version 3 Improvements

NFS Version 3 introduced a variety of improvements to NFS performance and scalability. These features are covered in detail in the following sections.

Write Throughput

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

The NFS Version 3 protocol offers a better alternative to increasing write throughput by eliminating the synchronous write requirement while retaining the benefits of close-to-open semantics. The NFS Version 3 client significantly reduces the number of write requests it makes to the server by *collecting* multiple requests and then writing the collective data through to the server's cache. Subsequently, it submits a commit request to the server that causes the server to write all the data to stable storage at one time. This feature, referred to as safe asynchronous writes, vastly reduces the number of write requests to the server significantly improving write throughput.

The writes are considered *safe* because status information on the data is maintained indicating whether or not it has been stored successfully. Therefore, if the server crashes before a commit operation, the client will know by looking at the status indication whether to resubmit a write request when the server comes back up.

Read Throughput

Read throughput can be defined as the amount of time applications wait for file data to become available subsequent to issuing a read request. Both NFS Version 2 and Version 3 clients utilize local disk caching and read-ahead to enhance read throughput. In addition, NFS clients also maintain the cache after a file is closed. This is because in the common case where a file is reopened, read requests will often be satisfied by data that already resides in the cache. Together, these features help ensure that the data an application wants to read will be in the cache in advance of demand reducing waiting time and thus increasing read throughput.

Reduced Requests for File Attributes

Because read data can sometimes reside in the cache for extended periods of time in anticipation of demand, clients must check to ensure their cached data does not become invalid if a change is made to the file by another application. Therefore, the NFS client periodically acquires the file's attributes, which include the time the file was last modified. Using the modification time, a client can determine whether or not its cached data is still valid.

Keeping attribute requests to a minimum makes the client more efficient and minimizes server load, thus increasing scalability and performance. Therefore, NFS Version 3 was designed to return attributes for all operations. This increases the likelihood that the attributes in the cache are up to date and thus reduces the number of separate attribute requests. This is an improvement over NFS Version 2, which does not always return attribute information.

Efficient Utilization of High Bandwidth Network Technology

NFS Version 2 has an 8 KB maximum buffer size limitation, which restricts the amount of NFS data that can be transferred over the network at one time. In NFS Version 3, this limitation has been relaxed enabling NFS to construct and send larger chunks of data. This allows NFS to more efficiently utilize high bandwidth network technologies, such as FDDI, 100baseT Ethernet, and the SP Switch, and has contributed substantially to NFS performance gains.

Reduced Directory "Lookup" Requests

A full directory listing (such as `ls -l`) requires that name and attribute information be acquired from the server for all entries in the directory listing. NFS Version 2 clients query the server separately for the file and directory names list and attribute information for all directory entries in a lookup request. However, with NFS Version 3, names list and attribute information is returned at one time offloading both client and server from performing multiple tasks.

8.1.13 How NFS v3 and TCP Work Together

NFS relies upon lower layer transport protocols to transmit data over the network. NFS was originally designed to use UDP as the transport protocol for communication over both local- and wide-area networks. UDP was chosen because, in the past, it provided better performance than TCP. Although UDP works very well for LANs, it has some limitations when used for communication over WANs. One drawback is that because it is an unreliable protocol, UDP is not designed to handle the special problems introduced by high-latency, low-bandwidth network connections.

Now that there are high-performance TCP implementations available, NFS has been enhanced to utilize TCP.

Because there is more overhead required to set up an initial connection using TCP than there is using UDP, all NFS client traffic can be multiplexed over one TCP connection to the server. In other words, NFS clients use one connection to each server regardless of the number of client mounts per server. This keeps overhead to a minimum, uses fewer resources (sockets, descriptors, and so on), and makes recovery from failure faster. It also allows servers to scale in order to support larger numbers of NFS clients.

NFS Version 3 works in conjunction with TCP to provide further performance gains. Unlike NFS Version 2, NFS Version 3 has no fixed limit on the amount of data that can be transferred between client and server in a single read or write request. The client and server are able to negotiate whatever transfer size they can both support. In general, the larger the transfer size, the more efficient TCP becomes because more data can be sent at one time.

Backward Compatibility with NFS UDP

Considering these benefits, it is easy to see why TCP is expected to become the preferred transport protocol for NFS, but it will take time for the installed base of UDP clients and servers to switch over. Therefore, in order to provide backward compatibility, NFS clients and servers can be implemented so that they can support both protocols. For example, when an NFS connection is in the process of being established, the client and server negotiate which protocol to use based on what each one supports.

8.2 Virtual Shared Disk Tuning

IBM Virtual Shared Disk (VSD) is a subsystem that allows data in logical volumes on disks physically connected to one node to be transparently accessed by other nodes. VSD only supports raw logical volumes, not journalled file systems. See Figure 34 for an illustration of a simplified virtual shared disk implementation.

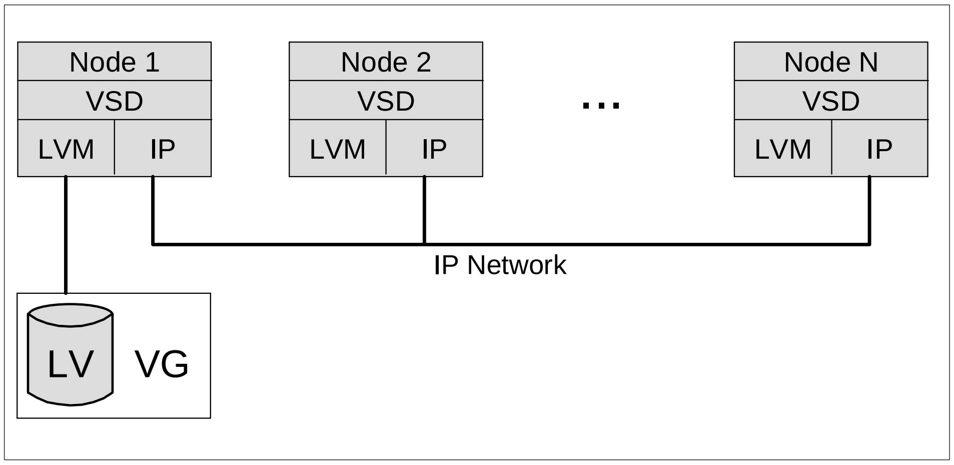


Figure 34. Virtual Shared Disk Implementation

Using the VSD software, applications that are running on different nodes (belonging to the same partition) can access a raw logical volume as if it were local at each of the nodes.

When the logical volume is local for a specific node, this node is known as the server node. The other nodes (the client nodes) can access this same logical volume because the VSD software routes the I/O requests to the server node and sends the results back to the client nodes.

The following information is excerpted from VSD tuning information in the VSD manuals. It serves as a quick reference to tuning and performance of VSD but does not address everything about configuration or setup information. If you need more information, please refer to the VSD documentation.

8.2.1 Tunable Parameters Related to VSD

The main VSD tunable areas are:

- Logical Volume Manager (striping and other characteristics)
- IP communications adapter (usually the switch)
- Virtual Shared Disk cache buffer
- Buddy buffer
- Maximum I/O request size
- Request blocks
- pbufs
- mbufs

The actual processing capability required for Virtual Shared Disk service is a function of the application I/O access patterns, the node model, the type of disk, and the disk connection.

If you plan to run time-critical applications on a Virtual Shared Disk server, remember that servicing disk requests from other nodes might conflict with the demands of these applications.

Make sure that you have sufficient resources to run the Virtual Shared Disk program efficiently. This includes enough buddy buffers of sufficient size to match your file system block size, as well as sufficient rpoolsize and spoolsize blocks in the communications subsystem.

8.2.2 Logical Volume Manager Tuning Considerations

There is always an associated logical volume for every virtual shared disk defined and configured in a system. Every virtual shared disk I/O request eventually becomes an I/O request to the associated logical volume (unless you get a cache hit at the sever). This mapping of virtual shared disk I/O requests to the associated logical volume I/O request is handled transparently by the VSD subsystem. All the performance tuning considerations that apply to a logical volume also apply to its virtual shared disk.

8.2.3 SP Switch Considerations

For virtual shared, set the maximum IP message size (utilized by the virtual shared disk driver) to 61440 (60 KB). Ensure that the maximum buddy buffer size is 256 KB, since the value you assign to maximum buddy_buffer_size in the SDR also limits the maximum size of the request that the IBM Virtual Shared Disk subsystem sends across the switch. For example, if you have:

- A request from a client to write 256 KB of data to a remote virtual shared disk
- A maximum buddy buffer size of 64 KB

OR

- A maximum IP message size of 60 KB

the following transmission sequence occurs:

1. The IBM Virtual Shared Disk subsystem divides the 256 KB of data into four 64 KB requests in four buddy buffers.
2. Each 64 KB block of data becomes one 60 KB packet and one 4 KB packet for transmission to the server through IP.
3. At the server, the eight packets are reassembled into four 64 KB blocks of data each in a 64 KB buddy buffer
4. The server then has to perform four 64 KB write operations to disk and return four acknowledgments to the client.

A better scenario for the same write operation would use the maximum buddy buffer size of 256 KB:

- The same 256 KB client request to the remote virtual shared disk
- The maximum buddy buffer size of 256 KB
- The maximum IP message size of 60 KB

This produces the following transmission sequence:

1. The 256 KB request becomes four 60 KB packets and one 16 KB packet for transmission to the server via IP.
2. At the server, the five packets are reassembled into one 256 KB block of data in a single buddy buffer.
3. The server then performs one 256 KB write operation and returns an acknowledgment to the client.

The second scenario is preferable to the first because the I/O operations at the server are minimized. A perfect scenario would be one where the IBM Virtual Shared Disk component does not use buddy buffers at all - when the client request is less than or equal to the maximum IP message size. For example:

- A request from a client to write 60 KB of data to a remote virtual shared disk server

- A maximum IP message size of 60 KB

When you use the switch, send pool clusters are used instead of buddy buffers as long as the request size is less than ip_message_size, as in the example just cited. Buddy buffers are used only when a shortage in the switch buffer pool occurs or when the size of the request is greater than ip_message_size. If you see buddy buffer shortages, instead of increasing your buddy buffers, you need to increase your switch send pool size.

8.2.3.1 mbufs and the Switch Pool

mbufs are used for data transfer between the client and the server nodes by the IBM Virtual Shared Disk subsystem's own UDP-like Internet protocol. If you are using the switch (css0) as your communications adapter, the IBM Virtual Shared Disk component uses mbuf clusters to do I/O directly from the switch's send and receive pools.

If you notice that the indirect I/O statistic (from the IBM Virtual Shared Disk Perspectives Statistics notebook page or from the output of the `vsdstat` command) is incremented consistently, run errpt to check the error log. If you see the line:

```
IFIOCTL_MGET(): send pool shortage
```

you should consider increasing the size of the send and receive pools.

To check the current sizes of the send and receive pools, type:

```
# lsattr -l css0 -E
```

The default size for each pool is 524288 bytes (512 KB).

To change the sizes of the pools to 4 MB, type:

```
# /usr/1pp/ssp/css/chgcss -l css0 -a spoolsize=4194304
# /usr/1pp/ssp/css/chgcss -l css0 -a rpoolsize=4194304
```

These commands increase the send and receive pool size to 4 MB.

Note that you must reboot the node for the new sizes to take effect.

IBM suggests you allow 16 MB for mbufs and clusters. You can set this value by issuing:

```
# no -o thewall=16384
```

To see what your current system mbuf setting is, type:

```
# no -a | grep thewall
```

System performance considerations regarding mbufs and mbuf clusters also apply to virtual shared disk environments.

8.2.4 Buddy Buffers

The Virtual Shared Disk server node uses buddy buffers to temporarily store data for I/O operations originating at a client node to handle requests that are greater than ip_message_size. In contrast to the data in the cache buffer, the data in a buddy buffer is purged immediately after the I/O operation completes.

Note

Buddy buffers are used only when a shortage in the switch buffer pool occurs or on certain networks (for example, the Ethernet).

The values associated with the buddy buffer are:

- Minimum buddy buffer size allocated to a single request
- Maximum buddy buffer size allocated to a single request
- Total size of the buddy buffer

Buddy buffer space is allocated in powers of two. If an I/O request size is not a power of two, the smallest power of two that is larger than the request is allocated. For example, for a request size of 24 KB, 32 KB is allocated on the server.

If you are using the switch as your adapter for virtual shared disk, we recommend settings 4096 (4 KB) and 262144 (256 KB), respectively, for minimum and maximum buddy buffer size allocated to a single request.

To define the total size of the buddy buffer, consider the remote I/O throughput for the server, and specify the number of maximum-sized buddy buffers in the buffer. For example, if you expect the server to serve 10 MB per second on behalf of remote clients, and a request spends an average of 60 milliseconds on the server, multiply 10 MBps by 0.06 seconds and, for safety, double or triple the result for a total buddy buffer size of 1.8 MB (eight 256 KB maximum buddy buffers).

If the virtual shared disk statistics consistently show queued requests waiting for buddy buffers, do not add more buddy buffers. Instead, increase the size of the switch pool or spread the data over disks attached to other nodes to prevent the bottleneck.

Note

If your application uses the fastpath option of asynchronous I/O, the maximum buddy buffer size must be greater than or equal to 128 KB. Otherwise, you will get EMSGSIZE *Message too long* errors.

8.2.5 VSD Buffer Allocation

Your application should put all newly allocated buffers on a page boundary. If your I/O buffer is not aligned on a page boundary, the VSD device driver will not parallelize I/O requests to underlying virtual shared disks, and performance will be degraded.

8.2.6 The Cache Buffer

Each VSD device driver, that is, each node, has a single cache buffer shared by cacheable virtual shared disks configured on and served by the node. The cache buffer is used to store the most recently accessed data from the cached virtual shared disks (associated logical volumes) on the server node. The objective is to minimize physical disk I/O activity. If the requested data is found in the cache, it is read from the cache rather than the corresponding logical volume.

Data in the cache is stored in 4 KB blocks. The content of the cache is a replica of the corresponding data blocks on the physical disks. Write-through cache semantics apply; that is, the write operation is not complete until the data is on the disk.

When you create virtual shared disks with VSD perspectives or the `createvsd` command, specify the *cache* option or the *nocache* option. IBM suggests that you specify nocache (or make the cache buffer small) in most instances (especially in the case of read-only or other than 4 KB applications) for the following reasons:

- Requests that are not exactly 4 KB and not aligned on a 4 KB boundary will bypass the cache buffer but will incur the overhead of searching the cache blocks for overlapping pages.
- Every 4 KB I/O operation incurs the overhead of copying into or out of the cache buffer as well as the overhead of moving program data from the processor cache due to the copy.

- There is overhead for maintaining an index on the blocks cached.

If you are running an application that involves heavy writing followed immediately by reading, it might be advantageous to turn the cache buffer on for some virtual shared disks on a particular node. Choose the appropriate size of the cache based on the expected throughput and the expected time lag between writes and reads. For example, if the expected throughput is 100 4 KB-aligned I/O operations per second and reads lag writes by 0.5 seconds, calculate the cache buffer size by multiplying 100 x 0.5 and, as a safety factor, double this for a total of 100 cache blocks.

The `lsvsd -s` command gives detailed statistics on virtual shared disk cache hits and I/O activities. This tells you which virtual shared disks are heavily used. See “VSD Statistics” on page 118 for information on how to interpret VSD statistics.

8.2.7 Maximum I/O Request Size

The following factors limit the block size that the virtual shared disk (VSD) subsystem uses to process each request:

- The largest block size the VSD subsystem uses is the smaller of max_buddy_buffer_size or 256 KB.
- If the VSD uses the switch as its adapter, the maximum max_IP_msg_size that can be sent is 65024 bytes (63.5 KB). IBM suggests the value 61440 (60 KB) for the VSD device driver when css0 is defined as the VSD adapter in the SDR. The `ctlvsd -M` command can override the default. max_IP_msg_size should be set to a value that is a multiple of 512 bytes and is less than or equal to 63.5 KB (when the switch is used) and less than or equal to 24 KB (when the switch is not used). The `statvsd` command displays the current value.
- Setting max_IP_msg_size to more than 24 KB when using communication adapters with small MTU can overflow the adapter driver’s internal buffer causing the IP layer to drop packets. This forces the virtual shared disk device driver to retry (sometimes without success) resulting in a timeout.

Every virtual shared disk I/O request is subject to both limits. For example, with max_buddy_buffer_size of 262144 (256 KB) and max_IP_msg_size of 61440 (60 KB), if an application requests a single 64 KB read to a virtual shared disk served by the local node, the request is passed down to the local logical volume as one 60 KB request and one 4 KB request.

The atomicity of an I/O operation is gated by the size of the virtual shared disk request, rather than the size of the application request. If the virtual shared disk request is smaller than the application request, the application request will be split down to the size of the virtual shared disk request.

8.2.8 Request Blocks

The number of request blocks is the total number of physical I/O operations that have been issued by all processes on a node but have not completed. The number includes requests to both local and remote devices. Because large requests may be broken up into smaller subrequests, the request block number may be several times bigger than the actual number of pending read/write requests.

For example, if I/O requests are:

- Issued at a rate of 1000 I/O operations per second
- Broken on the average into three pieces
- Responded to in 50 milliseconds

the algorithm for calculating the number of request blocks is 1000 X 3 X 0.05 for a total of 150 request blocks. You may want to increase the number somewhat as a safety precaution to account for the possibility of workload surges. Specifying an inordinately large number of request blocks can have a negative performance impact. However, a large number of request blocks can flood the network causing servers to run out of mbufs and causing unnecessary retransmissions. What constitutes a large number of requests depends on how large the request size is and how many nodes are in the system.

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 of request blocks infinitely, queuing would occur elsewhere in the operating system.

The number of request blocks can be set and changed with the VSD perspectives graphical interface or by using the `vsdnode` and `updatevsdnode` commands, respectively.

8.2.9 Virtual Shared Disk pbufs

Buffers called pbufs are used for actual physical I/O requests that are submitted to the disks. A pbuf shortage affects overall performance negatively. However, you must also be careful not to exceed your environment’s limitations.

pbufs are specified as a way of controlling the number of pending device requests for a specific virtual shared disk on its server node. pbufs are specified on a per-device basis.

You can set the number of pbufs to be allocated per virtual shared disk by the Designate as an IBM VSD node... action, the rw_request_count parameter of the `vsdnode` command, or the SMIT vsdnode_dialog fast path.

Each virtual shared disk, regardless of its activity and whether the node is a client or a server, is allocated these pbufs. Each pbuf is 128 bytes long. You can calculate how much of the kernel heap you need for pbufs using the following formula:

heap_allocated = nvsd x nreq x 128

where:

nvsd is the number of virtual shared disks

nreq is the number of pbufs you are requesting

Make sure that heap_allocated never exceeds the available kernel heap. For example, given that one eighth of the heap is available for pbufs (about 32,000,000 bytes), and 1300 virtual shared disks are configured, the value of nreq cannot exceed 192.

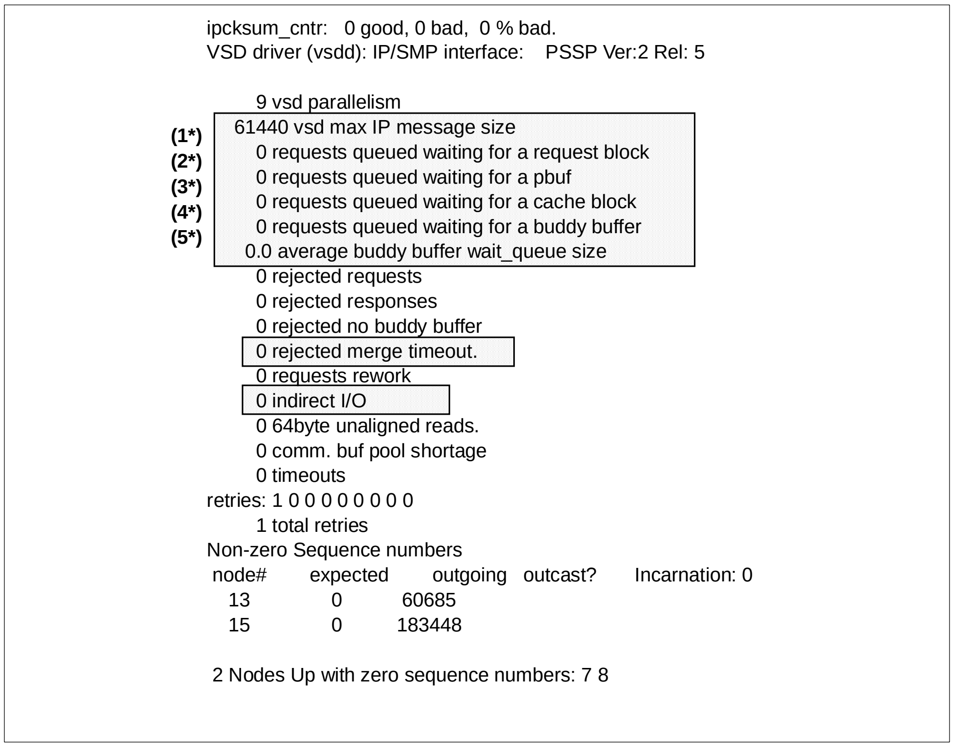
To check on the interactions among request blocks, pbufs, and cache blocks using the VSD perspectives graphical user interface, do the following:

- With a node selected, click **Action** -> **View and Modify Properties...**
- Select the VSD Node Statistics page.
- View the statistics and decide which parameters to tune.

The `statvsd` command also gives detailed statistics on request shortages, pbuf shortages, and cache block shortages. You can run the command together with lsvsd -s before and after an application execution to determine how to tune these parameters to best fit a particular workload.

8.2.10 VSD Statistics

To list VSD usage statistics, you can use either the perspective's graphical user interface or the `statvsd` command.



```
          ipcksum_cntr:   0 good, 0 bad,  0 % bad.
          VSD driver (vsdd): IP/SMP interface:    PSSP Ver:2 Rel: 5

                9 vsd parallelism
(1*)        61440 vsd max IP message size
(2*)            0 requests queued waiting for a request block
(3*)            0 requests queued waiting for a pbuf
(4*)            0 requests queued waiting for a cache block
(5*)            0 requests queued waiting for a buddy buffer
              0.0 average buddy buffer wait_queue size
                0 rejected requests
                0 rejected responses
                0 rejected no buddy buffer
                0 rejected merge timeout.
                0 requests rework
                0 indirect I/O
                0 64byte unaligned reads.
                0 comm. buf pool shortage
                0 timeouts
          retries: 1 0 0 0 0 0 0 0 0
                1 total retries
          Non-zero Sequence numbers
           node#       expected       outgoing   outcast?      Incarnation: 0
             13            0          60685
             15            0         183448

           2 Nodes Up with zero sequence numbers: 7 8
```

Figure 35. statvsd Command Output

Issue the `ctlvsd -V` command to reset the statistics.

Issue the `statvsd` command to show the statistics.

The statistics provide information about the number of logical read and write operations, the number of remote logical read and write operations, the number of client logical read and write operations, the number of physical reads and writes, the number of cache hits for read, and the number of 512-byte blocks read and written.

The number of blocks read and written is cumulative; so, issue the `ctlvsd -V` command to reset this count before measuring it. The local logical operations are requests that were made by a process executing at the local node, whereas the remote logical operations were made by a process executing on a remote node. Client operations are those local logical requests that cannot be satisfied locally and have to be sent to a remote node. Physical operations are those server operations that must be passed to the underlying disk device. Cache read hits are those server reads that do not require a device read because the read operation was satisfied from the IBM Virtual Shared Disk cache.

These statistics show requests queued waiting for buddy buffers. If we are using the switch adapter, we do not have to add more buddy buffers, but we have to increase the switch send pool or spread the data over disks of other nodes.

If using the switch, the VSD uses pool segments for I/O directly from the switch's send and receive pools.

The output shown in the figure provides statistics. The ones in the first highlighted area are useful to set the parameters properly in the SMIT panel that is shown later in these student notes.

The following are the main fields to check:

- VSD *parallelism* (default value is 9 and should always be used) indicates how VSD divides the I/O requests into smaller units.
- *max_IP_message* is the value set in the last field of the SMIT panel. The value is set to 61440 (60k) because, in the example defined on the test machine, VSD was using the switch (see the second field in the SMIT panel). In fact, 61440 is the value that must be defined when using the switch while the default value is 24576 (24k).
- In the SMIT panel the *VSD Request Count* is set to 256, which is the recommended number of outstanding VSD requests. This number should be changed if the statistics show that some requests are queued waiting for a block.
- If the number of requests waiting for a pbuf is not 0, change the Read/Write Request Count field, which specifies the maximum number of outstanding requests the VSD device driver allows for each underlying logical volume.
- The VSD device driver implements an optional write-through cache with a block size of 4 KB. The recommended values are the ones displayed in the SMIT panel (the minimum is a 256 KB = 64 x 4 KB cache, and the maximum is 1 MB = 256 x 4 KB), but, if we have not defined the maximum as 256 and see that there are requests waiting for cache blocks, we can tune this value.
- We must consider the number of max-sized buddy buffers and their maximum size. These two fields must be checked if we get a number that is not zero for the requests waiting for a buddy buffer. The recommended value, when the switch is used as an IP adapter, is no more than 4, which

means 1024 KB buddy buffer size in combination with the recommended maximum size of 256 KB.

To change the characteristics of the VSD analyzed by the statistics, we can use the notebook page of spvsd GUI (perspectives) or SMIT.

8.2.11 Tuning Virtual Shared Disk Performance

The IBM Virtual Shared Disk device driver passes all its requests to the underlying Logical Volume Manager subsystem. Before you tune the virtual shared disk, check that the I/O subsystem is not the bottleneck. If an overloaded I/O subsystem is degrading your system's performance, tuning the virtual shared disk will not help. In the case of I/O subsystem overload, consider spreading the I/O load over more disks or nodes.

For best performance, do the following:

1. Use the defaults when defining virtual shared disks.
2. Turn IBM VSD caching off if you are not using your system for online transaction processing.
3. Do a performance run to collect statistics on the virtual shared disks, your I/O subsystem, and the CPU on all nodes (or use Performance Monitor to collect information during normal system operation). Issue statvsd several times during the performance run, and compare the values for the various statistics. Use iostat to check your disk utilization. If you notice increasing numbers of queued requests, do the following:
    - If the system is I/O bound (meaning your disks are more than 50 percent utilized), add disks.
    - If the system is CPU bound, add nodes or spread the workload on the virtual shared disk server nodes. You can use the Hashed Shared Disk data striping subsystem to spread the workload.
    - If nodes are doing excessive swapping due to insufficient pinned memory, which is used by pbufs, buddy buffers, cache, and the switch pool, reduce cache size.
    - If requests are queuing because of a shortage of buddy buffers or pbufs, you might have disk bottlenecks. Spread the data or add disks to the server nodes.
    - If your application issues requests that are larger than 64 KB, set your maximum buddy buffer size to 256 KB.

  - If you see too many retries, check for disk bottlenecks. If that is not the problem, consider increasing the switch pool size (see “mbufs and the Switch Pool” on page 113).

4. Reset the statistics counter by running the `ctlvsd` command (you can use the Run Command... action of the IBM Virtual Shared Disk Perspective graphical user interface).
5. Do another performance run.

You should generally operate with IBM Virtual Shared Disk caching off. Memory is better allocated to the operating system itself, for paging, and to the cache belonging to the application using the virtual shared disk. To turn IBM Virtual Shared Disk caching off, do the following:

1. Shut down applications that use virtual shared disks.
2. If you do not use the IBM Recoverable Virtual Shared Disk subsystem, unconfigure the virtual shared disks.
3. Select one or more nodes.
4. Use the **Run Command...** action and run the `updatevsdtab` command to change the cache/nocache option to nocache.
5. If you do not use the IBM Recoverable Virtual Shared Disk subsystem, configure the virtual shared disks.
6. If you do use the IBM Recoverable Virtual Shared Disk subsystem, refresh the virtual shared disk configuration.
7. Restart your applications.

If you use caching, remember that the IBM Virtual Shared Disk component only caches 4 KB requests aligned on 4 KB boundaries.

8.2.12 Virtual Shared Disk Tuning Recommendations

Here are some rules of thumb (especially for GPFS) for tuning the VSD subsystem:

- Use a switch send and receive pool size of 16 MB on VSD servers.
- If your application uses the fastpath option of asynchronous I/O, the maximum buddy buffer size must be greater than or equal to 128 KB, otherwise you will get EMSGSIZE *Message too long* errors.
- Set the buddy buffer count to at least two times the number of disks on VSD servers.
- Set the buddy buffer count on clients to **1**.

- Set the buddy buffer size to 256KB.
- Set max_IP_msg_size to 61440 to maximize the switch packet size.

8.3 General Parallel File System Tuning

The following General Parallel File System Performance Tuning information is an excerpt from the GPFS documentation. It only addresses performance and tuning of GPFS. For more general GPFS information, consult the full GPFS documentation.

IBM General Parallel File System for AIX (GPFS) provides file system services to parallel and serial applications running on the RS/6000 SP. GPFS allows users shared access to files that may span multiple disk drives on multiple SP nodes. For an overview, see Figure 36 on page 123.

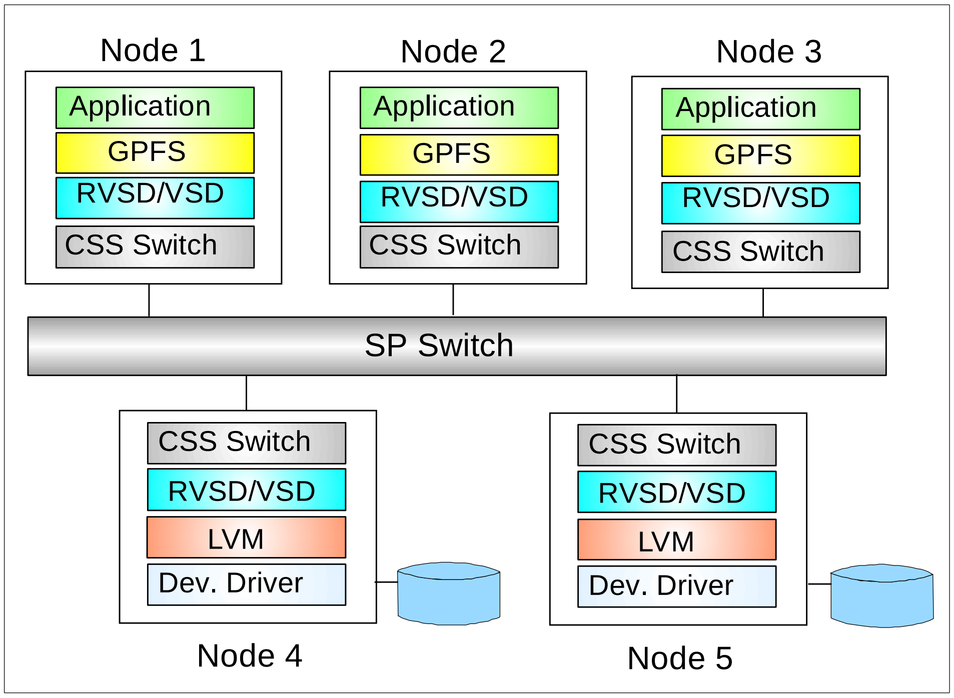


Figure 36. GPFS Overview

GPFS allows parallel applications simultaneous access to the same files or different files from any node in the configuration while managing a high level of control over all file system operations. It offers extremely high recoverability while maximizing data accessibility.

Using GPFS to store and retrieve your files can improve system performance by:

- Allowing multiple processes or applications on all nodes of the SP system simultaneous access to the same file using standard file system calls.
- Increasing aggregate bandwidth of your file system by spreading reads and writes across multiple disks.
- Balancing the load evenly across all disks to maximize their combined throughput. One disk is no more active than another.
- Supporting large amounts of data allowing you to have bigger file systems.
- Allowing concurrent reads and concurrent writes to files in the file system (this is very important for parallel processing).

GPFS builds on the shared disk concept at the heart of the Virtual Shared Disk component of the Parallel System Support Programs for AIX by taking advantage of the speed of the SP Switch and using it to accelerate parallel file operations that would overload serial file system management.

8.3.1 Planning for GPFS

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 immediately useful file system.

GPFS configuration requires you to specify several operational parameters that reflect your hardware resources and operating environment. Later, during file system creation, you have the opportunity to specify additional parameters based on the expected size of the files. These parameters define the disks for the file system and how data will be written to them.

8.3.2 Configuration Considerations

Configuration involves defining the nodes to be included in the GPFS subsystem and specifying how they operate. You can provide a list of nodes as input to configuration or allow GPFS to configure all the nodes in your SP system. You also have the option of installing a sample configuration file with preset values that you can accept as defaults or modify to suit your needs, or you can create and install a configuration file of your own.

8.3.3 Estimating Node Count

When creating a GPFS file system, overestimate the number of nodes that will mount the file system. This input is used in the creation of GPFS data structures that are essential for achieving the maximum degree of parallelism in file system operations. Although a larger estimate consumes a bit more memory, insufficient allocation of these data structures can limit node ability to process certain parallel requests efficiently, such as the allotment of disk space to a file. If you cannot anticipate the number of nodes, allow the default value of 32 to be applied. Specify a larger number if you expect to add nodes. During configuration, you can specify two parameters that control how much cache is dedicated to GPFS. These values can be changed later; so, experiment with larger values to find the optimum cache size that improves GPFS performance without affecting other applications. In order for changed values to take effect, you must restart GPFS.

Note

The sum of these two parameters must not exceed 80 percent of real memory.

pagepool	This is the size of the cache on each node. It can range from a minimum of 4 MB to a maximum of 512 MB per node. This value must be specified with the character M, for example, 80M. The default is 20M.
mallocsize	This area is used exclusively for GPFS control structures. It can range from a minimum of 2 MB to a maximum of 128 MB per node. This value must be specified with the character M, for example, 8M.The default is 4M.
maxFilesToCache	This is the number of i-nodes to cache for recently used files that have been closed. Storing a file's i-node in cache permits faster re-access to the file. The default is 200, but increasing this number may improve throughput for workloads with high file reuse. Increasing it where file reuse is not great wastes kernel heap storage. Do not increase this parameter if you are in doubt. Each cached file requires space in the mallocpool for an i-node plus approximately 800 bytes of metadata. GPFS will not use more than 50 percent of the mallocpool for this purpose. If you increase maxFilesToCache, you should also increase the mallocsize parameter to a value equal to 2 x (maximum i-node size for the file system + 800) x maxFilesToCache. For example, to set maxFilesToCache to 1000 for a file system with a maximum i-node size of 4096, mallocsize should be approximately 10 MB.

The only configuration parameter that directly addresses performance is priority. Sometimes called scheduling priority, this parameter sets the UNIX priority for the GPFS daemon. The default is 40. Priority increases with lower values.

8.3.4 GPFS Use of Virtual Shared Disks

GPFS accesses data using the facilities of the Recoverable Virtual Shared Disk (RVSD) Licensed Program and the Virtual Shared Disk (VSD) component of the Parallel System Support Programs for AIX, which allows application programs executing on different SP nodes to access a logical volume as if it were local at each node. RVSD, which allows a secondary or backup server to be defined for such a logical volume, is required even when there are no twin-tailed disks that can be physically accessed from more than one node because it provides fencing capabilities that preserve data integrity in the event of certain failures. *Managing Shared Disks,* SC23-4839 contains installation, management, and usage information for both VSD and RVSD.

Proper planning for GPFS installation provides:

- Sufficient processing capability for VSD servers to deliver data to clients
- Sufficient disks to meet the expected I/O load
- Sufficient connectivity (adapters and buses) between disks and VSD servers

8.3.5 Switch Tuning for GPFS

Use the dsh command to propagate the following rpoolsize and spoolsize values to all nodes:

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

Note

Do not set the rpoolsize and spoolsize values on the Control Workstation, since it does not have a switch adapter card.

Ignore any message of the type ...poolsize != default.

You can use SMIT or the `mmchconfig` command to change the following configuration attributes after the initial configuration has been set:

- pagepool
- mallocsize
- priority
- autoload
- client_ports
- server_port_number
- server_kprocs

Items 1 through 3 take effect the next time GPFS is started. Items 4 through 7 require that the nodes be rebooted before new values take effect.

You must provide a descriptor for each GPFS IBM VSD to be created or passed to the GPFS file system. Each descriptor contains the following positional parameters for its IBM VSD.

8.3.6 GPFS Performance Tuning

GPFS performance depends on the correct specification of its parameters as well as the correct tuning of the functions it uses. The following information describes the parameters that affect performance. For most workloads, the parameters that have the greatest impact on performance are the file system block size and the amount of memory allocated to GPFS. There are other parameters that affect specific workloads.

File System Block Size
GPFS offers three block sizes for file systems: 16 KB, 64 KB, and 256 KB. Once you have set this parameter at file system creation, you cannot change it without recreating the file system. The 256 KB block size is the best choice for file systems that contain large files accessed in large reads and writes. This block size allows the most effective I/O operations. The 16 KB block size optimizes use of disk storage at the expense of large data transfers and should be used for other types of applications. The 64 KB block size offers a compromise. It makes more efficient use of disk space than 256 KB while allowing faster I/O operations than 16 KB. You should choose the block size based on the application set that you plan to support.

8.3.7 Additional GPFS Considerations

There are several other parameters that are set during GPFS configuration. In most environments, the default values result in efficient performance. Some environments, however, can benefit from adjustment.

Priority
Much of the GPFS code runs as a multithreaded daemon. By default, the priority of the daemon is 4. You may wish to increase the priority of the daemon if your users execute a large number of time-critical I/O operations.

Token Management
The token management parameters control simultaneous file access. Do not change these settings unless you are familiar with AIX kernel and thread concepts.

The number of client ports and server processes controls the number of token operations that can be performed concurrently. The default settings should be adequate for workloads that are not very demanding on locks. If your applications are doing large amounts of very fine-grained write sharing of files, you might consider increasing these values.

client_ports
This is the number of communication ports that are reserved for requesting file access tokens.

server_kprocs
This is the number of kernel processes that are dedicated to processing file access token requests. These processes await token requests, and their number determines how efficiently multiple parallel processes can access a single file.

server_port_number
Two consecutive UDP ports are defined starting at this port number. The token manager receives token requests through these ports. The value of this parameter must be the same for all nodes in the GPFS configuration.

The number of these processes begins with the value specified as server_port_number and continues for the value specified as server_kprocs. server_port_number must be the same for all nodes in the GPFS configuration.

8.3.8 GPFS Performance and Scaling

This section describes some performance figures that prove how scalable GPFS is. The performance measurements were mainly done to find upper and lower bounds for aggregate bandwidth of GPFS. The results are based on GPFS Release 1.1, RVSD Release 2.1.1 and PSSP 2.4.

It should be mentioned that the test program is not representative of most *real* workloads in that it does nothing but I/O; there is virtually no processing time between read and write requests. Few applications will show such behavior except, possibly, for short periods. For this reason, it is likely that GPFS can support more clients than were tested.

The tests were run with 256 KB block size. The rpool and spool for the switch were 16 MB. As you can see in Figure 37, there is nearly a linear increase in throughput when the number of servers is increased. For example, with eight clients utilizing one server, we got an average read throughput of 56.6 MBps. When increasing the number of servers to four, we got an average throughput of 215.2 MBps.

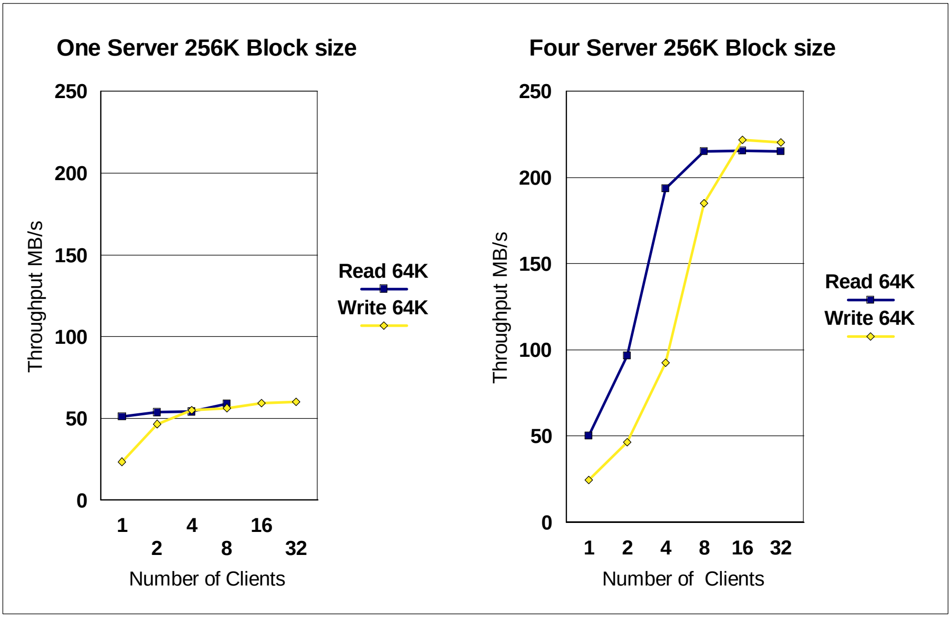


Figure 37. GPFS Performance and Scaling on a 332 Mhz SMP Node

Table 12 gives a more detailed overview of the same test results.

Table 12. GPFS Performance Test with 64 KB Read Requests

Clients	1 Server (MBps)	2 Servers (MBps)	4 Servers (MBps)	8 Servers (MBps)
1	51.2	55.3	50.2	52.7
2	53.7	98.5	96.5	109.7
4	54.1	107.6	193.6	208.9
8	59.0	117.1	215.2	367.1
16	N/A	N/A	215.5	365.1
32	N/A	N/A	215.3	362.7

8.3.9 Applications and Performance

The pattern of I/O generated by the application set is a major factor in the determination of GPFS performance. The following I/O patterns can be described:

Applications that do sequential reads or writes in increments of the file system block will see the highest throughput, and the use of larger file system blocks will maximize this. The limiting factor in these cases is either the number of disks that are available or the adapters and buses that attach the disks. The file system aids the performance in this case by doing read-ahead of blocks that will be used soon and by overlapping the actual write of data to a disk with the progress of your application.

Applications that do sequential writes using smaller blocks that create a new file or extend an existing one should see throughput that is only slightly worse than full block writes. The additional system calls to deliver the same amount of data will consume more CPU and that processing is the only difference. The actual disk I/O is overlapped with continued application processing as was the case for full block writes.

Applications that do sequential reads smaller than a block size will have disk characteristics similar to full block I/O because the file system does the same read ahead. Throughput in this case will be somewhat slower than full block size operations because of the additional CPU required when less data is transferred per call.

Applications that do small sequential writes that overlay data in existing files will cause additional I/Os because the file system must merge the changed portion of the block into the existing portion of the block. This causes the first write into a block to fetch the existing block from disk. Subsequent writes to the same block from the same node use the same copy unless there is an intervening write from another node. GPFS will write the block when all data is written and overlap it with further application processing.

Applications that do random reads of files are gated by the speed of a single disk. In this case, the file system cannot prefetch the data and must do disk reads synchronously with the application read call. Applications that do random writes of full blocks should see performance similar to sequential writes if sufficient page pool space is allocated. Applications that do random writes of odd sizes experience the time required to fetch the relevant sectors from disk and modify the contents. This slows these operations to the speed of a single disk.

Parallel applications that read the same file read at the speed of the disk subsystem for each instance of the application. Disk subsystems that do caching in the disk head or in the path to the disk provide significant benefit to this class of application.

Parallel applications that write to the same file should attempt to write large contiguous regions of the file from individual instances of the application rather than writing small elements interleaved with elements from other instances of the application. The larger regions assigned to each node allow a coarser more efficient lock operation and more efficient use of the disks. A node that is processing a contiguous I00 MB set of writes can obtain a set of tokens covering the region and hold it without interference. The disk sectors that back up that region are also void of interference from other nodes. A node that is processing 100-byte interleaved writes must obtain tokens for the disk area containing the data and give up those tokens when a write occurs within the same data block. As part of processing these writes, the entire disk sector must be read and written back as part of token relinquish. Parallel applications should be designed to write the largest contiguous region that is consistent with the needs of the application from a single node.

Chapter 9. Common SP Performance Problems

When tuning an SP system, several other problems can cause lower-than-expected performance. The following sections shed more light on these problems, how to detect them, and what tuning parameters to change in order to alleviate them.

9.1 The Nagle Algorithm

A problem that often occurs on the SP system is that an application runs very slowly when using the SP Switch, while performance is significantly better on an Ethernet or FDDI network interface. This problem is sometimes caused by the Nagle Algorithm (used by TCP/IP) interacting with the delayed ACK (acknowledgment) timer.

Nagle algorithm

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

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

Following are the specific rules used by the Nagle Algorithm for deciding when to send data:

- If a packet is equal to or larger than the segment size (or MTU), send it immediately.
- If the interface is idle, or the TCP_NODELAY flag is set, and the TCP window is not full, send the packet immediately.
- If there is less than 1/2 of the TCP window in outstanding data, send the packet immediately.
- If sending less than a segment size, and if more than 1/2 the window is outstanding, and TCP_NODELAY is not set, wait up to 200 msec for more data before sending the packet.

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

With the rfc1323 parameter not set to 1, and having a large segment size for the network, sending full IP packets can cause a 5 packet/sec rate. Table 13 lists the window size where this occurs for various network segment sizes.

Table 13. TCP/IP Pacing Degradation Window

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

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

```
netstat -I css0 1
```

The output will show the switch and total IP packet rates per second as shown in Figure 38 on page 135.

```
# netstat -I css0 1
    input   (css0)     output           input   (Total)    output
 packets  errs  packets  errs colls  packets  errs  packets  errs colls
  125696     0   110803     0     0   356878     0   287880     0     0
     119     0      216     0     0      123     0      221     0     0
     117     0      222     0     0      120     0      224     0     0
     115     0      225     0     0      117     0      227     0     0
     115     0      202     0     0      117     0      204     0     0
     115     0      207     0     0      117     0      209     0     0
     116     0      201     0     0      118     0      203     0     0
     115     0      211     0     0      118     0      213     0     0
```

Figure 38. netstat -I Command

The following are suggestions on how to avoid the Nagle algorithm:

1. If you are running an application and do not have access to the source code, use the no command to increase the TCP window.

   Increasing the TCP window may not always be effective because increasing the tcp_sendspace and tcp_recvspace sizes on the sending and receiving nodes may cause other negative effects on the SP system or to other applications running on the system. Make sure that you set rfc1323 to **1** if the window size exceeds 65536.

2. Change the MTU size of the switch.

   Changing the MTU of the switch moves the window and buffer size combination where the 200 msec delay is invoked. When writing 32 KB buffers to a TCP connection, if the TCP/IP window is 65536, only 5 packets/sec are transmitted. If you change the MTU of the switch interface to 32768, there is no delay on transmitting a 32768 buffer because it is the same size as the segment size of the switch. However, reducing the MTU of the switch to 32768 degrades the peak TCP/IP throughput slightly. Reducing the MTU even further degrades the peak throughput even more.

3. From within an application, you can increase the TCP window size on the socket by using the SO_SNDBUF and SO_RCVBUF settings on the socket.

   - For good performance across a switch, we suggest that both SO_SNDBUF and SO_RCVBUF be set to at least 524288 on both the client and server nodes. You need to set both sides since TCP uses the lowest common size to determine the actual TCP window.
   - If you set the SO_SNDBUF and SO_RCVBUF sizes larger than 65536, you need to set TCP_RFC1323 also on the socket unless the no options already set it. Setting TCP_RFC1323 to 1 takes advantage of window sizes greater than 65536.

    - You also want to ensure that the system setting for sb_max is at least twice the TCP window size or else sb_max will reduce the effective values for SO_SNDBUF and SO_RCVBUF.

4. Set TCP_NODELAY on the socket of the sending side to turn off the Nagle algorithm.

   All data that is sent will go immediately, no matter what the data size. However, if the application sends very small buffers, you will significantly increase the total number of packets on the network.

One common problem that causes unexpected Nagle behavior is setting tcp_sendspace and tcp_recvspace to a high number and forgetting to set rfc1323 to 1 on all nodes. In addition, setting tcp_sendspace and tcp_recvspace large on one node and not on the other nodes can cause Nagle to occur.

On systems where a node is talking to several other nodes, it is harder to see the Nagle effect. In this case, the only way to detect it is to examine iptraces to extract a single socket's traffic. What can happen is that if one node is talking to two nodes, each connection can be seeing the Nagle effect, and the packet rate over the switch is 10 packets/sec. If you have one node talking to five nodes, the packet rate can be 25 packets/sec but the aggregate switch traffic 1.5 MB/sec. This rate exceeds the throughput on a slow Ethernet network, but is well below what the switch can handle.

9.2 External Server Considerations

Connections through external networks to other machines can restrict performance on certain nodes on the SP system. For example, some external servers may not support rfc1323 and, therefore, can only use a maximum TCP/IP window of only 65536 to transfer data to or from a node. This causes performance problems with connections to these servers to behave differently from other connections on the same node to other SP nodes.

In some cases, external servers have to communicate with additional external servers. When such external servers establish a connection, the negotiated tunables may be set to small values. For example, the TCP/IP window size value is set to the least common denominator. In the case of an external server having a tcp_recvspace of 16384, that is the TCP/IP window size used. Such a small TCP/IP window size provides slow throughput if the sending node is on the far side of a switch from a gateway node.

If traffic is routed through other external network routers, you may not be able to use the optimal maximum transmission unit (MTU) or tcp_mssdflt size for that network. If this happens, adjust the tcp_sendspace and tcp_recvspace values accordingly.

To find the optimal tcp_sendspace and tcp_recvspace sizes, get the largest MTU size that the router will handle to other networks of the same type. To get the optimal tcp_sendspace and tcp_recvspace sizes for single-stream transfers, use the formula shown in Figure 39.

```
t = m * q

where:

t =  Optimal tcp_sendspace and tcp_recvspace sizes

m  = Largest MTU size that the router will handle to other networks

q   = Smaller of the transmit queue and receive queue size for the adapter
```

Figure 39. Calculating tcp Send/Receive Space Sizes

This does not apply to the RS/6000 SP switch because the switch adapter does not have transmit and receive queues. See “Switch Adapter Pools” on page 79 for more information on tuning rpoolsize and spoolsize.

The number produced by the formula in Figure 39 is the largest that you can use for a single socket before the adapter drops packets and TCP/IP has to do a retry on the dropped packets. If you have more than one active socket, the size calculated using this formula needs to be divided by the number of active sockets. You want to avoid dropping packets on the adapter to get optimal throughput performance.

Some of these problems cannot be overridden using tunables on the node but need to be considered when tuning a node that communicates to outside servers. On AIX 4.2.1, there are two new tunables that help in solving the small packet problem to external servers. These *no* settings are:

- tcp_pmtu_discover
- udp_pmtu_discover

By setting these tunables to **1**, when a connection is established to a connection on a remote network (if the external server or workstation supports MTU path discovery), the connection will determine the largest segment size that it can send without fragmentation. This eases the problem of setting tcp_mssdflt to a compromise value.

9.3 Single-Server Multiple-Client Node Problems

Some application configurations, such as NFS, consist of a parent or server node with multiple client or remote nodes. Such applications have a potential for the client nodes (through a large TCP window size) to send large volumes of data to one server node using the nonblocking switch network, whereas the server node cannot handle the total traffic from the client nodes, due to demands on the server's mbuf pools. The dropped packets will be reflected in a large number of failures in netstat -m on the server node.

To further illustrate, if you had 64 client nodes, as shown in Figure 40 on page 139, with a TCP window size of 512 KB, the server node would need buffer space of 32 MB just to accommodate the incoming packets from the client nodes, all other connections and traffic aside. Remember, the maximum amount of space the can be allocated to thewall is 64 MB. To determine the server node mbuf requirements, get the number of client nodes that will simultaneously send data to the server node, multiply by the TCP window size on each client node, then again multiply by the number of sockets that each client opens on the server. If the server has multiple roles, add additional mbuf space as required. You can see how easy it can be to exceed the maximum amount of memory that can be allocated for the network (thewall) or even the limitations of the server node itself, which may only contain 128 MB of memory. To prevent this scenario from happening, you must reduce the combined amount of data arriving from the client nodes to the server nodes.

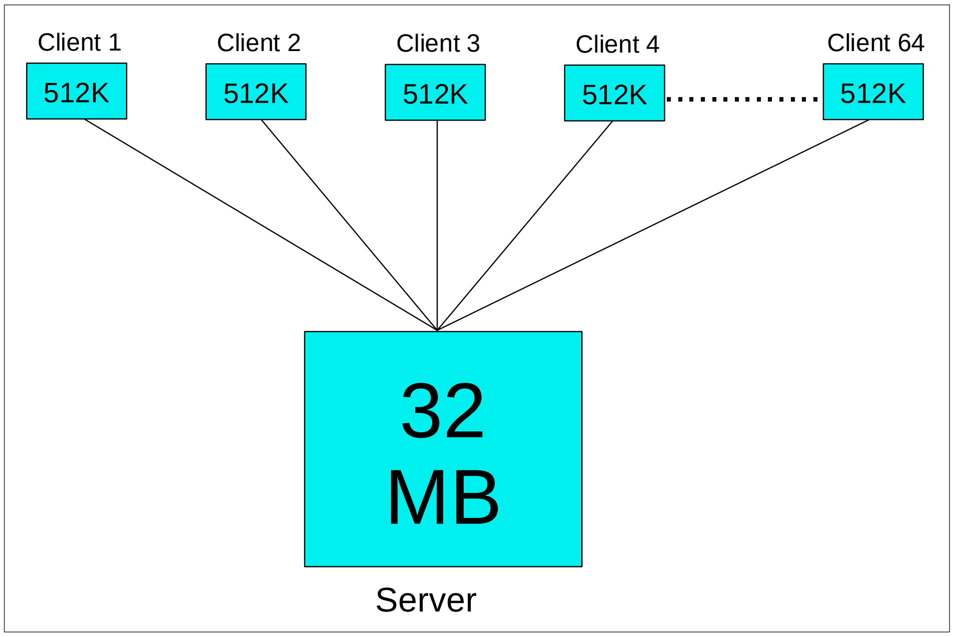


Figure 40. Single-Server Multiple-Client Scenario

You can use two methods to accomplish this:

The first is to try to restrict the client nodes by setting tcp_sendspace and tcp_recvspace small enough so that the combined data sent by the client nodes does not exceed the buffer space on the server. While this reduces the receive buffer space required on the server, if that traffic must then be redirected to a terminal or remote file, you need to double the mbuf requirements to accommodate sending it back out again. If there are other applications on the client nodes that require the high switch bandwidth, they will not get the same throughput and may suffer due to the smaller setting for tcp_sendspace and tcp_recvspace. Traffic will back up, and applications may slow down. It is an administrator's tradeoff for large data transfers on the client versus mbuf allocation on the server. To determine what values you should assign to tcp_sendspace and tcp_recvspace, first select the maximum number of buffers to allocate in the servers' mbuf pool, divide by the number of client nodes, and multiply by the average message size to get your window size. Set tcp_send and tcp_receive on both the client and server side.

The second method would be to restrict tcp_recvspace on the server node. The TCP window is negotiated upon establishment of the connection; the window value will be set to the lesser of tcp_sendspace and tcp_recvspace and will only impact client-to-server connections.

If at all possible, the best solution would be to set the socket window sizes from within an application leaving the tcp_sendspace and tcp_recvspace settings alone This can be done by using the setsockopt() call within the application to change the values for SO_SNDBUFF and SO_RCVBUFF. Changing these values affects only the socket that the setsockopt() call was made against. All other connections would use their own setsockopt() call settings or the tcp_sendspace and tcp_recvspace settings.

Keep in mind that UDP has no mechanism for windowing and can have a greater amount of outstanding I/O on the connection. UDP should not be used to send large amounts of data over the switch because you could deplete the servers' mbuf pools very rapidly. Also, be aware of very small messages. The minimum allocation from the mbuf pool is 256 bytes; so, you can chew up all the mbuf space with a small amount of data.

An example of how a typical program is modified to handle the setsockopt() call is shown in Figure 41:

```
/*We are the client if transmitting*/
if(options) {
   if(setsockopt(fd,SOL_SOCKET, options, &one, sizeof(one)) <0
                                err("setsockopt");
}
if(nodelay) {
   if(setsockopt(fd,IPPROTO_TCP,TCP_NODELAY,&one, sizeof(one)) <0
                                err("nodelay");
}
if(rfc1323) {
   if(setsockopt(fd,IPPROTO_TCP,TCP_RFC1323,&one, sizeof(one)) <0
                                err("rfc1323");
}
if (setsockopt(fd,SOL_SOCKET,SO_SNDBUF,&window,sizeof(window)) <0)
                        err("setsendwindow");
if (setsockopt(fd,SOL_SOCKET,SO_RCVBUF,&window,sizeof(window)) <0)
                         err("setreceivewindow");
if(connect(fd,&sinhim, sizeof(sinhim) ) < 0)
                err("connect");
mes("connect");
```

Figure 41. Sample setsockopt() Call

9.4 Gateway or Router Node Problems

Gateway or router nodes direct traffic between external networks and the SP system. Two different types of nodes are used as gateways: an existing SP node acting as a gateway, and the SP Switch used as direct-attached SP Router node, also known as the GRF router node.

If a lot of traffic is routed through a router node, it affects any other job using that node. We suggest that router or gateway nodes not be assigned to high priority or fast response time parallel jobs unless there are no other nodes available, or the amount of expected network traffic is small.

When tuning a router or gateway, you need to plan on enough buffer space in the node to handle traffic on multiple interfaces. Traffic from the SP Switch uses the send and receive pools. Traffic to Ethernet, Token Ring, and FDDI uses the system mbufs, while ATM uses its own buffer areas. Having a large difference between the switch pools and the amount of space for other adapter traffic leads to bottlenecks or dropped packets. The best initial settings are for the same amount of space for both sets of interfaces.

To optimize the packet size sent to remote networks if you are running AIX 4.2.1 or higher, you must set the variable tcp_pmtu_discover = **1**. This lowers the overhead on the gateway or router node. However, watch out for networks with more than a couple of hundred hosts. By turning tcp_pmtu _discover on, you are in effect creating a route in your routing table to every host that is out there. Any network greater than a couple of hundred hosts becomes very inefficient and performance problems arise. Be sure to turn this variable on if your network has the correct number of hosts.

9.5 Tuning the Control Workstation

Various models of network adapters can have different values for transmit queue sizes as shown in the following table:

Table 14. Control Workstation Network Adapter Queue Settings

Adapter Type	Queue Setting
PCI Adapter	256
MCA adapter AIX 4.2.1 or later	512

You can set these values using SMIT or the chdev command. If the adapter you are changing is also the adapter for the network you are logged in through, you will have to make the changes to the databases only. Then, reboot the Control Workstation (CWS) for the changes to become effective. Here is the command:

```
-P -l ent0 -a xmt_que_size=512
```

1. You must then reboot CWS in order for the changes to take effect.
2. To determine that the Transmit Queue Size has been changed to the proper value, issue the command:

```
lsattr -E -l adapter_name
```

where adapter_name= ent0.

9.5.1 Change Control Workstation Maximum Default Processes

When you first install your system, the number of processes is set to an AIX default. You will not be able to continue installing your system with this default value. The value must be increased. IBM suggests changing the maximum to 256. You can do this either with the SMIT panels or at the command line. The command is:

```
chdev -l sys0 -a maxuproc='256'
```

9.5.2 Change the Control Workstation Tunables

When you first install your system, the network tunable values are set to AIX defaults. Your system may not run efficiently with these values. Use the `no` command to display these values. When installing PSSP on your CWS, change the network tunables on the CWS to the values suggested in Table 15.

Table 15. Initial Control Workstation Parameters

Tunable	Recommended Initial Value
thewall	16384
sb_max	163840
ipforwarding	1
tcp_sendspace	65536
tcp_recvspace	65536
udp_sendspace	32768
upd_recvspace	32768
tcp_mssdflt	1448

When you change a network tunable value, it takes effect immediately. However, it is not preserved across a boot. To make the changes to the tunables effective across boots, add the `no -o` commands you used to change the network tunables to the bottom of the file /etc/rc.net. Remember that this is different from the nodes in the SP cluster. The dynamic tuning changes for the nodes need to be done in the /tftpboot/tuning.cust file.

9.6 ARP Cache Tuning

Address Resolution Protocol (ARP) translates Internet addresses for IP into unique hardware MAC addresses for all adapters on local links to a node. If a MAC address is not in the ARP cache, then, when transmitting to a remote adapter, an ARP broadcast is sent requesting the hardware address. These addresses are kept as a series of entries in buckets. The size and number of buckets are configurable. Configurations larger than 150 nodes in your RS/6000 SP environment could have performance problems due to the size of the ARP cache.

The variable arptab_nb specifies the number of ARP table buckets that can be stored in a node's ARP cache at any one time. This variable can be modified with the `no` command and is set in the /etc/rc.net script. The default value of 25 is not sufficient in environments with more than 150 nodes.

The second variable, arptab_bsiz, specifies (ARP) table bucket size. The default value is 7. The last ARP variable, arpqsize, specifies the maximum number of packets to queue while waiting for ARP responses. The default value is 1. The arpqsize value is increased to a minimum value of five when path MTU discovery is enabled. The value will not automatically decrease if path MTU discovery is subsequently disabled. This attribute applies to AIX Versions 4.1.5, 4.2.1, and later. The arpqsize variable is a runtime loadable attribute.

The relevant no command options are shown in Table 16.

Table 16. Default ARP Parameters in AIX.

Parameters	AIX System Defaults	Definition
arptab_nb	25	Number of buckets.
arptab_bsiz	7	Number of entries in each bucket.

Calculating ARP entries

The total available ARP entries are calculated using the variables:

arptab_nb * arptab_bsiz.

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

9.6.1 Updating the ARP Cache Size

The no commands to change the ARP cache size should be placed in /etc/rc.net, right after the first line in the file. If the changes are not placed at the beginning, and HACMP is installed, the changes may not take place. Figure 42 shows how the /etc/rc.net file should look after changes have been made.

```
#!/bin/ksh
/usr/sbin/no -o arptab_bsiz=10
/usr/sbin/no -o artab_nb=64
```

Figure 42. ARP Customizations

9.6.2 Determining the ARP Tuning Settings

For fast lookups, a large number of small buckets is ideal. For memory efficiency, a small number of medium buckets is best. Having too many buckets wastes memory (if the arptab_nb size were set to 128, bucket numbers above 66 would rarely be used). The recommended way to calculate the values for ARP cache sizing follows.

For systems with greater than 150 nodes, round the number of nodes down to the next power of 2, and use that for arptab_nb. Table 17 shows these values for systems from 1 to 512 nodes.

Table 17. Determining ARP Tuning Settings Based on the Number of Nodes

Number of nodes	arptab_nb value
1-64	25(default)
65-128	64
129-256	128
257-512	256

For nodes that have more than three network adapters, set arptab_bsiz to twice the number of active IP interfaces. Table 18 lists the sizes of the arptab_bsiz value based on the number of IP interfaces.

Table 18. Determining ARP Tuning Settings Based on Number of IP Interfaces

Number of interfaces	arptab_bsiz Value
1-3	7
4	8
5	10
6	12
7	14
8 or more	2 x number of interfaces

9.6.3 Detecting ARP Thrashing

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

9.6.4 ARP Cache Problem Determination

- Systems with more than 150 nodes can suffer ARP cache thrashing.
- Starting with PSSP2.3, Topology Services uses both the switch and SP Ethernet for heartbeating.
- ARP cache thrashing shows us the ARP cache containing a large number of entries. Issue the following command to determine how many ARP entries there currently are:

  ```
  arp -a | wc -l
  ```

- Changes to the ARP cache settings *must* be made in rc.net. Making changes using the `no` command has no effect on a running node but increases the pool size needed or can cause exhaustion of the pools.

Chapter 10. ADSTAR Distributed Storage Manager (ADSM) Tuning

When running ADSM on the SP system, the performance of a backup or restore is affected by several tunable settings. However, depending on the backup destination or restore source, the tunables and their settings are different. The following section lists the settings to initially try for running the ADSM server on MVS over an Escon channel and running the ADSM server on an SP system node. Because ADSM is not always the only application running on a client node or server, some of these values may not be reasonable in your environment.

There are up to four places where various tunables can be set to optimize the performance of ADSM. The following sections list these places and the initial values to use for ADSM over a switch, as well as ADSM over Escon to MVS.

10.1 SP Client Node Network Tunables

Table 19 shows the *no* tunable values that will achieve the best ADSM performance on the client nodes.

Table 19. ADSM Client Node Tuning Parameters

Parameter	Switch	Escon to MVS
thewall	16384	16384
sb_max	1310720	1310720
rfc1323	1	1
tcp_mssdflt	32768	32768

If you already have a larger value for thewall, you need to keep it.

10.2 SP Client Node ADSM Tunables

Table 20 shows the entries to use in the /usr/lpp/adsm/bin/dsm.sys file on the ADSM client nodes.

Table 20. ADSM Client Configuration File

Parameter	Switch	Escon to MVS
TCPWindowsize	256	64
TCPBuffsize	32	32
txnbyte	25600	25600
tcpnodelay	Y	Y

10.3 SP ADSM Server Node Tunables

Table 21 shows the *no* tunable values that will achieve the best ADSM performance on the SP ADSM server node.

Table 21. ADSM Server Node Tuning Parameters

Parameter	Switch	Escon to MVS
thewall	16384	N/A
sb_max	1310720	N/A
rfc1323	1	N/A
tcp_mssdflt	32768	N/A

If you already have a larger value for thewall, you need to keep it. Table 22 specifies the proper parameter entries to use in the /usr/lpp/adsm/bin/dsmserv.opt file on the SP system ADSM server node.

Table 22. ADSM Server Configuration File

Parameter	Switch	Escon to MVS
TCPWindowsize	256	N/A
TCPBuffsize	32	N/A
Txnbyte	25600	N/A
tcpnodelay	Y	N/A

10.4 Escon Gateway Node Tunables

Table 23 shows the *no* tunable values that will achieve the best ADSM performance on the node acting as the gateway across Escon.

Table 23. Escon Gateway Node Tuning Parameters

Parameter	Switch	Escon to MVS
thewall	N/A	15384
sb_max	N/A	131072
rfc1323	N/A	N/A

If you already have a larger value for thewall, you need to keep it. You will also need to set the MTU size of the Escon interface to 4096. To do that, you can issue the following command on the node with the Escon interface:

```
ifconfig es0 -mtu 4096
```

10.5 MVS ADSM Server Tunables

Table 24 shows the tunable settings on MVS that will get you peak throughput either across a switch or across a single Escon channel to MVS. Your actual throughput will be determined by the speed of the media being backed up to or restored from the SP system server or MVS server.

Table 24. MVS ADSM Settings

Parameter	Data	Data
DATABUFFERPOOLSIZE	240	32768
LARGENVELOPEPOOLSIZE	240	8192

You will also need to set the segment size on the Escon interface to 4096. These changes are in the gateway statement in the TCP/IP dataset definition. Here is a part of a sample dataset definition:

```
...

GATEWAY

129.40.15.11 = ESCF00   4096  HOST
```

Part 3. Performance Tools

Chapter 11. IBM Performance Tools

In this chapter, we discuss the tools available in AIX that are commonly used in determining and fixing performance related problems. Our main emphasis in this book is the interpretation of the information provided by these commands and not their syntactical use.

For a comprehensive discussion of the RS/6000 performance tools and their use, refer to *RS/6000 Performance Tools in Focus,* SG24-4989.

11.1 Overview

Tuning an RS/6000 SP requires an understanding of the work load that the RS/6000 SP will perform. Once you have this understanding, tuning of the system can commence. In AIX, on the RS/6000 SP, a wide variety of tools are available to first identify and understand the work load and then to help set up the system environment so that it is as close as possible to the ideal execution environment for the work.

Refer to Figure 43 on page 154 for an overview of the tuning commands and how they affect the various subsystems.

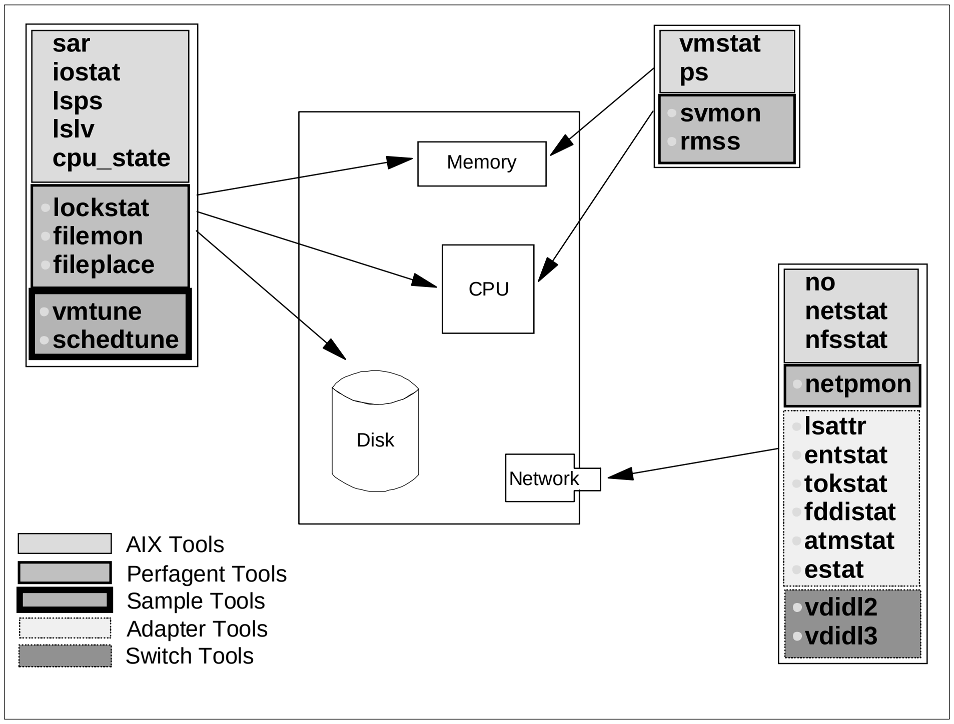


Figure 43. System Tuning Overview

The commands discussed in this chapter are provided as part of AIX. It is natural to assume that their use with the RS/6000 SP is the same as with the RS/6000 Symmetrical Multi-Processor (SMP) systems. This is not always the case. Often, we can adopt a more aggressive approach. An RS/6000 SP complex often requires several different tuning profiles. Each node or group of nodes performing different aspects of the work requires separate investigation. In a small number of cases, mainly when tuning an RS/6000 SP used for scientific research, the complex as a whole can be tuned using a single set of performance measurements and values.

Two categories to consider when using these commands on an RS/6000 SP are:

1. The nodes perform several different roles.

   The approach in this case is to compromise and trade the performance of one role off against another.

   We are more conservative when selecting values and choosing options. System tuning should not heavily bias one role at the expense of another of equal or greater importance. Less important work still needs consideration, which then often compromises the tuning with respect to key applications.

2. The nodes perform specific roles.

   More aggressive tuning options and values can be selected. Each node is tuned (and biased) towards the ideal execution environment for the specific role.

   We do not disregard the needs of other nodes. The tuning of a node cannot, for example, disregard the need to cooperatively share the network or a shared drive. Nor can the tuning neglect the primary reason the node is performing the task/role. For example, a Network File Server cannot be tuned exclusively for file management because it must, after all, serve those files to others across the network.

In a commercial environment, as the size of the complex increases, the nodes that make up the complex often take on specific roles.

With scientific applications, we typically see less mixing of applications in the complex. The complex as a whole is often aggressively tuned to run a single application. Tuning these systems requires a greater understanding of the application. For example, a large scientific application often performs a series of calculations, which is followed by a communications phase, and the cycle repeats. There should be a compromise between tuning for raw computation and network throughput. One phase may be biased at the expense of others but not to the exclusion of all others.

As the application roles for each node or group of nodes become specific, tuning becomes easier because:

- Templates can be built with predetermined tuning options.
- Determining causes and rectifying performance are simplified.
- The complex operates closer to its ideal performance levels with respect to a particular application.
- A segregated work load is easier to understand.

A final point before we discuss the tools available: AIX, PSSP, and any other tools or packages are overhead. Always tune the system to reduce their resource requirements.

11.2 Managing Memory Resources

Memory is a valuable and critical resource. Insufficient memory or poor use of memory results in serious performance problems.

The tools in this section are used to identify memory-related performance problems and to correct or minimize them.

11.2.1 Monitoring Memory with vmstat

This command provides statistical information collected by AIX for the Virtual Memory Manager (VMM), Central Processing Unit (CPU), and process scheduler.

In this section, we review aspects of this command that are associated with the VMM.

Use this command during periods when the system work load is representative of the system's expected work load. In some cases a system has several work load patterns. It is important to gain an understanding of memory utilization during these periods.

Example:

The output shown in Figure 44 was captured using `vmstat 5 10`. Using this command, we monitored the changing memory usage every five seconds for a total of 50 seconds. The first line is statistical information collected by AIX since the system was last booted.

```
# vmstat 5 10
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b    avm   fre  re  pi  po  fr   sr cy  in   sy   cs us sy id wa
 0  0 219963   201   0   0   0  14   50  0 138  457   65 13 10 59 18
 2  2 217167  3002   0   0   0  25   34  0 992 1638  238 30  8 49 13
 3  2 221754   228   0   0   1 402 1736  0 959 1198  217 31  8 56  5
 3  2 216648  5034   0   0   0  65   84  0 943 1383  194 31  7 60  1
 3  2 221020   536   0   0   0   0    0  0 948 2697  181 31 17 52  0
 3  2 222456   437   0   0   0 311  406  0 953 1730  187 35 12 52  0
 3  2 214221  8633   0   0   0  13   19  0 960 2955  224 25 14 58  3
 4  2 216900  5887   0   3   0   0    0  0 969 19824 381 31 17 48  4
 3  2 216366  5902   0   0   0   0    0  0 949 29553 193 30  9 59  2
 3  2 216054  4472   0   0   0   0    0  0 973 2570  274 26 11 46 17
```

Figure 44. vmstat Output

If the `fre` column (number of pages in the free list) is low (below 2 * MB of real memory - 8), and the `pi` column (page in rate/s) exceeds 5 per second, memory is overcommitted.

A high page scan (sr) to page steal (fr) ratio indicates that the memory subsystem is overactive. The higher this ratio, the more time the VMM is spending searching for available memory to allocate. Further investigation of this should be undertaken.

In AIX version 4 and onward, the page reclaim column (re) is always 0. Page reclaims (a page that is released by the VMM and then reclaimed by the same process before allocation to another process) are no longer recorded.

11.2.2 Monitoring Memory with sar

This command reports the values of the operating system activity counters, which are a quick and easy way to check on the system work.

Information for the VMM is shown by the paging activity counters.

Example:

Figure 45 on page 158 shows a system that is lightly loaded. The VMM has no difficulty fulfilling page requests. On average, 203 page faults per second were generated. The VMM maintained a large number of free memory slots throughout the time monitored. It was not necessary for the VMM to cycle through memory searching for free pages, and very little paging I/O was required.

If this system's performance was considered, we would conclude that memory is not a contributing factor to any problems.

```
# sar -r 1 10

AIX sp3en0 2 4 004008966700    09/29/98

00:25:01   slots cycle/s fault/s  odio/s
00:25:02   63326    0.00  248.18    1.82
00:25:03   63326    0.00    0.88    0.00
00:25:04   63325    0.00    0.91    0.00
00:25:05   63325    0.00    0.00    0.00
00:25:07   63529    0.00  212.73    3.64
00:25:08   63446    0.00  941.46    8.13
00:25:09   63333    0.00  526.32    4.39
00:25:10   63333    0.00    0.00    0.00
00:25:11   63333    0.00    0.00    0.00
00:25:12   63333    0.00    0.00    0.00

Average    63361       0     203       2
```

Figure 45. Monitoring Paging with sar

In Figure 45 on page 158, there is a 3-second peak in paging. This indicates an uneven work load distribution. In a heavily loaded system, these peaks need to be investigated. A multiuser environment needs a balanced work load to maintain a consistent response time for users. In our experience, users do not remember 999 subsecond responses. They remember the one that took 5 seconds.

11.2.3 Monitoring Memory with lsps

This command provides information about the paging space. Use it to check how much virtual memory is used. It is useful to know how much real memory is extended when considering a memory upgrade. If there is heavy paging, and memory is only extended by a small amount, additional memory is a very cost-effective upgrade.

Example:

Figure 46 on page 159 shows an example of the paging area for our system. We see that real memory is currently extended by 110 MB.

```
lsps -a
Page Space  Physical Volume   Volume Group    Size   %Used  Active  Auto  Type
paging01    hdisk2            rootvg         144MB      38     yes   yes    lv
paging00    hdisk1            rootvg         128MB      38     yes   yes    lv
paging00    hdisk2            rootvg          16MB      38     yes   yes    lv
```

Figure 46. Viewing Paging Space

The system paging area is spread across two drives. Paging to disk is occurring evenly across the two disks.

11.2.4 Monitoring Memory with ps

This command lists the current processes and their status. By examining the processes, we obtain an overview of how much memory each process uses. We also obtain information of the VMM overhead for each process.

This command takes a snapshot of the system showing a set of statistics for each process.

Example:

Figure 47 on page 159 is the output of `ps gvc` on our system. This example has been reduced to 10 lines.

```
# ps gvc
    PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
   31124      - A     0:01 1058  1332    96 32768    35    84  0.0  0.0 db2sysc
   31474 pts/13 A     0:05  704   516    52 32768   295    40  0.0  0.0 tcsh
   33146      - A     0:00    0   112   232 32768    49    80  0.0  0.0 xlC_r
  267064      ? A     0:00  185   384    12 32768   261     0  0.0  0.0 aixterm
  268156      - A     0:10    0 15116 17572 32768  2862  2408  4.0  2.0 xlCentry
  273814      - A     0:10   92  1036   148 32768   274   104  0.0  0.0 db2bp_s
  274866  pts/3 A     0:00    0   112   232 32768    49    80  0.0  0.0 xlC_r
  306432      - A     0:21 1507  1268   224 32768   485    16  0.0  0.0 dtwm
  306722 pts/19 A     0:00    3   176   232 32768   195   220  0.0  0.0 ksh
  307380  pts/3 A     0:10    0 12928 15384 32768  2862  2408  9.6  2.0 xlCentry
```

Figure 47. ps gvc Output

When using this command to review memory usage, examine:

- `PGIN`: Number of memory frames paged in
- `%MEM`: Percentage of system memory used

11.2.5 Monitoring Memory with svmon

This command shows the current state and usage of memory.

With the `svmon` command, memory page use can be viewed from the:

- System level
- Process level
- Segment level

11.2.5.1 Total Memory Usage

Always obtain an overview of memory usage from the system level. A performance problem caused by memory contention is unlikely when the system has sufficient memory available.

Example:

Figure 48 on page 160 is the output of `svmon -G` on our system.

```
# svmon -G
   m e m o r y          i n  u s e          p i n        p g  s p a c e
size  inuse free pin   work  pers  clnt  work pers clnt  size   inuse
65536 62724 2812 3508  41482 21242 0     3347 161  0     131072   25555
```

Figure 48. Global Memory View

Note

The values are displayed as memory pages. A memory page is 4069 bytes.

Interpretation of the `svmon` report:

- memory: System memory usage -
  size: Total size of real memory
  inuse: Amount of memory in use
  free: Amount of free memory
  pin: Pinned memory (memory pages that cannot be swapped out)

- inuse: Expands the column memory *in use*
  work: The system working set (data and stack regions)
  pers: Pages that are persistent on file.
  clnt: Client allocated memory (network clients)

- pin: Expands the column *pin*
  (Refer to preceding description of `inuse` columns)

• pg space: Size of the paging space
  size: Size of paging area
  inuse: Amount of page space in use (size of real memory extension)

11.2.5.2 Process Memory Usage

When memory has been identified as a performance issue, isolating the cause requires details on how memory is used by the processes.

Example:

Using `ps gvc | grep app` (refer to 11.2.4, “Monitoring Memory with ps” on page 159 for more detail on the use of this command), we identified a process running app (we are using app to represent an application). The process ID was 51994.

Figure 49 on page 161 is an example of using `svmon` to view a process’s memory usage.

```
# svmon -P 51994
   Pid                         Command        Inuse        Pin      Pgspace
51994                           find         4450        479         968

Pid:  51994
Command:  app

Segid  Type  Description          Inuse    Pin  Pgspace  Address Range
 380e  work  sreg[5]               1069    478      959  0..65535
 48d0  work  lib data                15      0        0  0..358
 4411  work  shared library text   3260      0        9  0..65535
 6b7e  work  private                 98      1        0  0..43 : 65304..65535
 62d8  pers  code,/dev/hd2:49266      8      0        0  0..7
```

Figure 49. Process Memory View

The app process has 4450 pages of memory allocated. A shared library, which this process uses, accounts for 3260 of these memory pages. Every additional copy of app, therefore, requires an additional 1190 pages of memory:

• Number of memory pages used minus the number of memory pages shared.

Further investigations of other processes executing on this system will determine whether the shared library is used by other applications. Using this information, we can determine whether the memory overhead for the shared libraries should be considered a system overhead or an application overhead.

Using svmon, memory usage can be classified as:

- System
- Application shared
- Application
- Application instance

This information is important for:

- Assessing a system’s memory requirements
- Evaluating application mixes for nodes
- Identifying candidate applications to be shifted

The Address Range column shows where in the allocated memory segment the process has referenced (the process’s memory footprint). This allows determination of how much free physical memory is required for the application to prevent paging.

11.2.6 Determining Memory Requirements with rmss

This command is used to limit the amount of real memory that the VMM will use. It does this by changing the value held in the kernel that determines the total amount of real memory installed.

Use rmss to determine the memory requirement of applications.

A careful and considered approach is required when using this tool. It sets the total available real memory that the system will use. This degrades system performance.

This command is used to determine how much real memory, at a minimum, is required by the application for reasonable performance (which is set by nonfunctional requirements). Often, more memory is better, and more memory certainly will not degrade an application’s performance. At some point, however, additional memory stops offering any real gain or ceases to be cost-effective.

Performance tuning means making the best use of what is available. Determining the minimum memory requirement will indicate if any significant gains can be achieved by further system investigation. If the system does not have the minimum amount of real memory required (which can be difficult to determine without testing each application’s memory requirement), memory must be increased, or the application’s memory requirements must be reduced.

Example:

```
rmss -c 128
```

This will change the amount of real memory that the VMM can use to a maximum of 128 MB. Use `rmss -r` to change the amount of real memory that the VMM can use back to the default value (real installed memory).

11.2.7 Tuning Memory with vmtune

This command changes the operational parameters of the virtual memory manager.

Changes to the VMM parameters can affect the system performance positively or negatively. Incorrect values can make the system unusable or even cause a system to crash.

The primary goal of using this command is to reduce paging.

The VMM maintains a list of free real-memory page frames (a page or page frame is 4096 bytes). Frames on this list are allocated to pages loaded into memory. The VMM attempts to ensure that a minimum number of real-memory pages are always available. The VMM will steal pages from running processes to maintain this list.

In an ideal environment, executing and terminating processes releases sufficient real-memory pages so that the memory requirements of other processes can be met without paging.

Changes made with *vmtune* should be placed in /tftpboot/tuning.cust. This will ensure that these changes will remain in effect when the system is booted.

Note

In the examples given, we have set the *vmtune* parameters to the default values for our installation of AIX 4.3.2.0.

11.2.7.1 Setting the Threshold Free List Value

When the amount of real memory available drops below a threshold value, page stealing starts. If this threshold value is too low, processes requesting memory will wait while the VMM steals and saves pages which it can then allocate. If the threshold value is set too high, active processes will have memory pages stolen when there are no impending requirements for additional memory. This also sets up an environment where memory thrashing is likely to be a problem.

With `vmtune`, the minimum number of real-memory page frames that must be maintained in the free list can be set. When the VMM replenishes the free list, it is set to a maximum size also set by `vmtune`.

To replenish the free list, VMM steals pages from other processes until the maximum level is reached. Above this point, normal memory management is resumed.

Example:

```
vmtune -f 120 -F128
```

Set the minimum free list size to 120 pages and the level at which page stealing is to cease to 128 pages. Ensure that the minimum value is less than the maximum value.

11.2.7.2 Setting Preferences Page Stealing

The VMM classifies memory as computational memory or file memory. Computational memory is memory which is part of the process working set (transitory - data regions and stack) or program code. File memory is all other memory.

When real memory drops below a defined threshold, both computational and file memory pages are stolen. Above a defined threshold, only file memory is stolen. Between the two threshold values, file memory is stolen unless the repage rate for file memory is high, in which case computational memory pages are also stolen.

The decision to set these parameters can only be made with an understanding of the workload. If the workload makes little reuse of file memory, decrease the point at which computational memory is stolen.

Example:

```
vmtune -p 30 -P 60
```

Both computational and file pages are stolen if the percentage of free memory is below 30 percent. Only file memory pages are stolen if the percentage of free memory is above 60 percent. Between these two values, the VMM will steal file memory unless file memory repage rates are high.

11.2.8 Tuning Memory with schedtune

This command changes the operational parameters for the VMM and process scheduler.

Changes to these parameters determine how AIX responds to memory thrashing. They also set the criteria that AIX uses to determine that memory thrashing is occurring.

Tuning with `schedtune` is done to smooth out infrequent peaks in load. The operational parameters tuned with this command are not intended to act in place of additional memory when the system's physical memory falls short of normal operating requirements.

In general, system tuning will not help when the system is short of memory.

Any system where memory is routinely overcommitted should have additional memory added, or the application's load/memory requirements should be reduced.

If, as a result of setting these parameters, thrashing is allowed to occur, the system may have a response perceived as better, but overall system throughput will have been reduced.

Place any changes made with `schedtune` into /tftpboot/tuning.cust. This ensures that, the next time the system is booted, the changes will remain in effect.

Note

In the examples given, we have set the `schedtune` parameters to the default values for our installation of AIX 4.3.2.0.

11.2.8.1 Memory Overcommitment

As part of load control, the scheduler suspends processes when memory becomes overcommitted. Memory is overcommitted once the threshold number of pages written to the paging space is exceeded, and this threshold value exceeds the number of page steals.

Example:

```
schedtune -p 4 -h 1
```

Memory is not considered overcommitted by the scheduler until four pages (p X h) are written to page space per second, and this is greater than the number of page steals per second.

By default, -h0 is used, which disables memory control.

11.2.8.2 Process Suspension

When memory is overcommitted, the scheduling algorithm suspends processes which repage (a page that belongs to the process and was reclaimed, and then soon afterwards is required by the process) at a greater rate than a threshold limit if this value exceeds the number of page faults for the process.

Example:

```
schedtune -r 16 -p 4
```

A process is not suspended until it repages 64 (r * p) memory pages, and this is greater than the number of page faults by the process.

11.2.8.3 Minimum Processes

When processes are suspended due to a memory overcommitment, a limit is set on the minimum number of processes that must be kept active. This is to prevent the system from being overly aggressive in suspending processes to eliminate a memory overcommitment situation.

Example:

```
schedtune -m 8
```

The scheduler always keeps at least 8 processes active even if memory is over committed.

11.3 Managing CPU Resources

Because the CPU is one of the fastest components in the system, it is rarely utilized 100 percent for an extended period of time (a few seconds at most). Often, when the CPU is 100 percent utilized by a program, the program is in an infinite loop.

Commercial applications that tie up the CPU in this way, even when they are not in an infinite loop, should be reviewed. In a multiuser environment, this is not acceptable.

Scientific applications often require very high CPU utilization during certain phases of execution. Workloads of this nature are normally run on dedicated nodes. In an environment where these programs are being run, and the nodes on which they execute have a mix of work, open a discussion with the user and/or systems administrator. Very few tuning options are effective in this situation. We have found rmuser to be most effective.

It is difficult to draw a line when reviewing CPU utilization. *If the percentage of CPU utilization exceeds x, we have a system constrained by CPU* is often a wild stab in the dark to provide a definition.

An understanding of the environment and the workload is required. A scientific application performing tight calculations is limited by the CPU at 100 percent utilization. Commercial applications will be limited below this point because they are not always able to utilize the CPU resource when it is available. The CPU requirement is unbalanced: at times processes are waiting on the CPU resource, and, at other times, the CPU is idle.

11.3.1 Monitoring the CPU with vmstat

We have already reviewed this command in 11.2.1, “Monitoring Memory with vmstat” on page 156.

vmstat is used to monitor CPU usage. It provides a single line report, which shows a quick status of CPU utilization.

Example:

Figure 50 is an example of using vmstat 5 on a system with free CPU resources. While monitoring this system, a peak in the workload occurs.

```
# vmstat 5
kthr     memory             page              faults        cpu
----- ------------ ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0 25565  5251   0   0   0   0    1   0 131  365  64  1  2 96  1
 6  0 41263   124   0   0   0  56   56   0 724 2564 450 41 24 34  0
 3  0 40600  1374   0   0   0  97  114   0 746 2097 393 40 21 38  0
 3  0 40769  1212   0   0   0   0    0   0 698 2142 389 40 24 36  1
 6  0 41019   871   0   0   0   0    0   0 719 2707 493 48 22 27  3
 0  0 41301   529   0   0   0   0    0   0 610 1439 304 25 10 64  0
 2  0 41517   479   0   0   0  64   71   0 628 2588 421 47 23 23  7
 6  0 43293   147   0   0   0 360  368   0 691 3429 407 74 26  0  0
 8  1 43636   190   0   0   0  86  114   0 724 3899 508 67 33  0  0
 6  0 44093   175   0   0   0 108  109   0 689 2807 378 71 29  0  0
 7  0 44416   415   0   0   0 120  120   0 721 3676 448 67 33  0  0
 7  0 44169   853   0   0   0  25   25   0 693 2927 403 74 26  0  0
 6  0 43075  2363   0   0   0   0    0   0 653 2907 436 85 14  0  0
 2  0 42395  3087   0   0   0   0    0   0 732 3413 432 80 17  2  1
 3  0 42635  2764   0   0   0   0    0   0 718 3020 415 59 19 19  3
 2  0 42748  2647   0   0   0   0    0   0 752 2704 432 55 16 29  0
 1  0 41626  4092   0   0   0   0    0   0 748 2506 452 52 17 21 10
 0  0 41228  4610   0   0   0   0    0   0 730 2806 450 54 17 26  3
 3  0 40773  5216   0   0   0   0    0   0 673 2782 445 67 21 12  0
```

Figure 50. CPU Monitoring with vmstat Output

Interpreting system activity from this `vmstat` report:

1. The CPU is lightly utilized.

   The idle average is 40 percent(id).
   The users average is 40 percent (us).
   The system average is 20 percent (sy).
   The low I/O wait is (wa).

   Conclusion: CPU utilization of 40 percent is not due to an I/O bound system.

2. The system work load peaks.

   The idle average is 0 percent.
   The user average is 75 percent.
   The system average is 25 percent.
   Low I/O wait.
   Free list low (fre).
   High number of page scans (sr).

High number of page steals (`fr`).

Conclusion: VMM is searching for memory to replenish the free list. This overhead is constraining the user application CPU requirement.

3. Work load constrained by CPU.
   The idle average is 0 percent.
   The user average is 85 percent.
   The system average is 15 percent.
   No I/O wait.
   Free memory available.
   No paging.

   Conclusion: The system is CPU bound. Users' CPU utilization increased as system CPU utilization decreased. This confirms the above conclusion that VMM constrained the user CPU utilization.

4. The system work load drops.

   the idle average is 20 percent.
   The user average is 50 percent.
   The system average is 20 percent.
   The I/O wait is 10 percent.

   Conclusion: The work load peak is over; a back log of work is being cleared in the I/O subsystem.

The peak in workload was a few lines on this report, each line representing 5 real-time seconds. The peak in workload therefore lasted 40 seconds. Once completed, the system response was still degraded while a backlog of work was cleared.

Further investigation of this will be required. Identify the program/application and determine its loading pattern (when and how often this application is executed).

11.3.2 Monitoring the CPU with time

The `time` command gives CPU and real-time execution figures for commands and applications.

Use this command to determine an application's CPU utilization. A good example of when it should be used is given in 11.3.1, "Monitoring the CPU with vmstat" on page 167. An application was executed which used substantial CPU resources. `time` can be used to quantify the CPU requirement.

Example:

Figure 51 shows the result of using `time` to measure the CPU resource requirement to execute our application apprep. This is an unobtrusive method of collecting CPU utilization for applications.

```
# time apprep

real  3m12.76s
user  2m38.72s
sys   0m26.54s
```

Figure 51. Checking CPU Utilization with Time

11.3.3 Monitoring the CPU Using ps

We discussed this command in 11.2.4, "Monitoring Memory with ps" on page 159. This command can also be used to obtain an overview of CPU usage by a process or group of processes.

Users tell us which programs take a long time to execute. This can be misleading when investigating CPU utilization. A slow program is often not a high user of the CPU. Use `ps` to determine which processes have high CPU utilization, and thereby assemble a candidate set of programs to investigate.

Example:

Figure 52 on page 170 is a sample script using `ps` to find the top 10 CPU users.

```
#!/bin/ksh

HEAD=`ps vc | head -n 1`

ps vc | head -n 1
ps gvc | grep -v "$HEAD" | sort +10 -r | head -n 10
```

Figure 52. Top 10 CPU Users Script

Using the script in Figure 53, we see a situation on our system where a number of processes have high CPU resource requirements.

.

```
# top10
    PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
  34910  pts/6 A     0:20    0   296   424 32768   197   228 48.8  0.0 apphog
  42064  pts/6 A     0:08    0   300   424 32768   197   228 29.6  0.0 apphog
  37196  pts/6 A     0:01    0   316   424 32768   197   228 25.0  0.0 apphog
  39602  pts/6 A     0:03    0   304   424 32768   197   228 20.0  0.0 apphog
  40112  pts/6 A     0:02    0   308   424 32768   197   228 18.2  0.0 apphog
  41028      - A     0:03   40   896  1184 32768    22    40  0.7  0.0 dtterm
  25044      - A     8:35   44   268    44 32768    73     0  0.6  0.0 rvsdd
  20488      - A     0:45  222   976   852 32768    22    40  0.5  0.0 dtterm
   7770      - A     0:51  153   900   700 32768    22    40  0.4  0.0 dtterm
   1562      - A     0:51  153   900   700 32768    22    40  0.2  0.0 dtterm
```

Figure 53. Top 10 CPU Users

Interpreting this example:

- The application apphog occupies the top 5 positions:

  The application is not paging.

  High CPU utilization (%CPU).

  Short execution time (TIME).

- Analysis of report:

  The application shows a high CPU utilization and was in execution for only a short period of time.

  In comparison to other programs, this one consumed a lot of CPU resources in a very short period of time.

  The program is not paging indicating the memory foot print is small and may only be within the CPU cache.

- Conclusion:

  This program is in a tight CPU intensive loop and may be in an infinite loop.

The application apphog needs further investigation.

11.3.4 Monitoring the CPU with sar

In 11.2.2, “Monitoring Memory with sar” on page 157, we discussed using this command to check paging statistics.

This command provides a simple method to review CPU utilization.

11.3.4.1 CPU Utilization Report

To obtain an overview of CPU utilization, that is, to determine how much time is spent doing actual work versus the time spent being idle or waiting on I/O, use the default `sar` options.

Example:

Figure 54 on page 172 is an example of using `sar` to report CPU utilization. The report is for a system where the CPU is a limiting factor and is 100 percent utilized. The top10 CPU report discussed in 11.3.3, “Monitoring the CPU Using ps” on page 170 should be used to find processes with high CPU resource requirements.

```
# sar 1 10

AIX sp3en0 3 4 000081007000    09/28/98

07:48:32    %usr    %sys    %wio   %idle
07:48:33      77      23       0       0
07:48:34      82      18       0       0
07:48:35      81      19       0       0
07:48:36      79      21       0       0
07:48:37      81      19       0       0
07:48:38      80      20       0       0
07:48:39      79      21       0       0
07:48:40      81      19       0       0
07:48:41      77      23       0       0
07:48:42      67      33       0       0

Average       78      22       0       0
```

Figure 54. CPU Utilization Report Using sar

11.3.4.2 CPU Queue Report

A controlling factor in CPU utilization for a system is an application’s ability to maintain runnable work. If work is always available to processors when they are free, the CPU’s full processing potential is used.

The scheduler maintains a run queue of work to be dispatched when a CPU is available. If the scheduler is unable to put work in this queue, CPU resources will be wasted. This is shown in the CPU idle or I/O wait time.

When CPU utilization is high, use `sar` to determine the reason:

- Is CPU utilization high due to efficient use by the application?
- Is CPU utilization high because the CPU is overloaded and we have a backlog of work?

Example:

The `sar` report shown in Figure 55 on page 173 is from a node performing CPU-intensive calculations. There are always six threads performing these calculations, and they compete for CPU time.

```
# sar -q 1 10

AIX sp3en0 3 4 000081007000    09/28/98

09:20:49 runq-sz %runocc swpq-sz %swpocc
09:20:50     6.0     100
09:20:51     6.0     100
09:20:52     6.0     100
09:20:53     6.0     100
09:20:54     6.0     100
09:20:55     6.0     100
09:20:56     6.0     100
09:20:57     6.0     100
09:20:58     6.0     100
09:20:59     6.0     100

Average      6.0     100
```

Figure 55. Monitoring CPU Queue Lengths with sar

From the report shown in Figure 55, we can determine the following:

1. The run queue size (`runq-sz`, the threads that are dispatchable), and the run queue utilization percentage (`%runocc`) over the 10 seconds monitored remained the same.

   Either the CPU is utilized by one thread with sufficient priority to keep the CPU allocated, or these threads are CPU-bound and remain dispatchable when forced to relinquish the CPU.

   It is possible that, as one process gave up the CPU, it became undispatchable and, at the same time, another process filled the vacant position on the dispatchable queue. We monitored the system for 10 seconds. It is unlikely that this situation would consistently occur over 10 seconds.

2. The dispatchable threads waiting to be paged in (swpq-sz) and the percentage of time the swpq-sz queue is occupied (%swpocc) in our example were 0.

   This indicates that the running threads were always busy, and, therefore, the VMM kept their pages active.

   As the VMM kept their pages in real memory, the amount of real memory in the node was adequate.

11.3.5 Monitoring the CPU with iostat

This command provides I/O statistics and is used to determine how effectively the CPU is utilized.

Indications that the CPU is not efficiently utilized and that I/O, particularly disk I/O, is the likely reason, are:

- CPU utilization is low.
- System performance is poor.
- The system is executing work.
- The CPU is forced to wait on I/O.

Isolating an I/O problem is covered in 11.4, “Managing Input/Output Resources” on page 179. In this section we limit the discussion of this topic to determining whether CPU utilization is low due to I/O.

Example:

Figure 56 shows a report collected from a lightly loaded system, using iostat -t. The system’s CPU resource utilization is below 40 percent (% user + % sys), and 6 percent of CPU time is spent waiting for I/O.

```
# iostat -t 1 10
tty:      tin        tout   avg-cpu:  % user    % sys     % idle    % iowait
          3.8        857.3             17.3      9.2       53.4      20.1
          0.0        159.6             17.9      7.6       67.6       6.9
          0.0         80.9             15.0     19.7       59.8       5.5
          0.0         80.8             14.0     24.4       55.2       6.4
          0.0         80.9             10.1     17.0       65.2       7.7
          0.0         80.9             11.4     14.7       67.3       6.6
          0.0         80.8             16.0      9.9       66.1       8.1
          0.0         80.9             14.0     12.1       67.4       6.5
          0.0         80.9             13.5     13.0       65.4       8.1
          0.0         80.9             14.7     10.6       66.8       7.9
```

Figure 56. Monitoring CPU I/O Waits Using iostat -t

In this example, I/O waits would not be considered a problem because the system CPU resource shows a significant amount of idle time.

I/O waits are a problem when the CPU has very little idle time, in which case CPU utilization is constrained by the I/O subsystem.

There are often I/O waits with a lightly loaded system. Since there are only a few processes executing, the scheduler is unable to keep the CPU utilized. Environments with few active processes and significant I/O waits require application changes. Changing an application to use asynchronous I/O is often effective.

As the number of processes executing on a system increases, it is easier for the scheduler to find dispatchable work. I/O waits therefore diminish.

Enable I/O pacing to reduce I/O waits if:

- The system has a significant number of processes.
- It is expected that the CPU should remain busy.
- There are significant I/O waits.
- A large volume of writes are issued by a few (often one) processes.

11.3.6 Checking Active CPUs Using cpu_state

This command sets and displays which processors will be active when the system is restarted. It is only available for SMP nodes (on uniprocessor RS/6000 SP nodes, the flexibility of allowing the CPU to be disabled was not deemed necessary).

With cpu_state, processors are selectively enabled or disabled. The changes take effect when the node is rebooted.

Generally, disabling processors reduces performance. We do not recommend disabling CPUs on production systems.

When assessing the performance of applications, there are situations when comparisons between an SMP and a uniprocessor are required, for example, to determine the performance benefits of an SMP node.

We used this feature to perform low-level application tracing. With multiple CPUs, it was difficult to paste execution streams into a sequenced execution flow.

Example:

Figure 57 on page 176 is a report from an 8-way SMP node after 7 of the 8 processors were disabled.

```
# cpu_state -l
        Name    Cpu      Status         Location
        proc0   0       enabled         00-0P-00-00
        proc1   1       disabled        00-0P-00-01
        proc2   2       disabled        00-0Q-00-00
        proc3   3       disabled        00-0Q-00-01
        proc4   4       disabled        00-0R-00-00
        proc5   5       disabled        00-0R-00-01
        proc6   6       disabled        00-0S-00-00
        proc7   7       disabled        00-0S-00-01
```

Figure 57. Active CPU Report

11.3.7 Managing CPU Usage with nice and renice

Use these commands to set thread priorities.

A lower nice value for a thread increases the importance of the thread. A normal nice value for a foreground process is 20, while a background process normally has a nice value of 24.

AIX adjusts the priority of threads that do not have a fixed priority according to the amount of CPU attention the process receives. AIX also adds to the nice value a base value according to the types of processes. A background process has a higher base value than a foreground process, for example.

Note

The AIX dispatcher and scheduler only support threads. When a process is started, a thread is also created. The thread performs work on behalf of the process. When altering a process's priority, its thread priority is effectively altered.

Example:

nice +10 apphog

Start *apphog* with a lower process priority. In this case, 10 more will be added to the default nice value:

renice +10 n (n is the process id)

Lower the priority of process *n* by 10.

11.3.8 Managing CPU Utilization with schedtune

In 11.2.8, "Tuning Memory with schedtune" on page 165 this command was used to alter the scheduling parameters.

In this section, we discuss the parameters that are set using *schedtune* to control the CPU resource allocation.

Note

- Place changes made using *schedtune* into /tftpboot/tuning.cust. This insures that, when the system is rebooted, the changes remain in effect.
- In the examples given, we have set the *schedtune* parameters to the default values for our installation of AIX 4.3.2.0.

11.3.8.1 Controlling Process Priority

For threads that do not have a fixed priority, the priority is dynamically adjusted. This is done by AIX in an attempt to give all processes a chance to receive CPU time.

AIX adjusts priority using a penalty system, as follows:

- By penalizing any process that is executing when the 10ms timer interrupt occurs:
  1. Add 1 to the CPU usage counter assigned to the process.

2. Calculate a penalty value by multiplying the process’s CPU usage counter by a definable penalty value and then dividing by 32.
3. Add the penalty value to the process nice value when determining the process priority.

- By decreasing a process’s penalties once a second by multiplying the CPU usage counter by a definable decay value and then dividing by 32.

Example:

```
schedtune -r 16 -d 16
```

The rate at which the penalty accumulates is 16 (-r 16). The penalty decay rate is 16 (-d 16).

Effectively, the accumulate and decay rates are 0.5 (16 / 32).

Table 25 shows an example. This is the cycle for a process that is executing on the CPU eight times when the 10ms interrupt occurs.

Table 25. Thread Priority Calculation

10ms Timer Interrupt	Usage Count	Penalty Value	Nice Value
Initial settings	0	0.0	60.0
1st 10ms interrupt	1	0.5	60.5
2nd 10ms interrupt	2	1.0	61.0
3rd 10ms interrupt	3	1.5	61.5
4th 10ms interrupt	4	2.0	62.0
5th 10ms interrupt	5	2.5	62.5
6th 10ms interrupt	6	3.0	63.0
7th 10ms interrupt	7	3.5	63.5
8th 10ms interrupt	8	4.0	64.0
1 sec interrupt	4	2.0	62.0

11.3.8.2 Controlling Timeslices

The CPU timeslice allocated to each dispatched thread is set using `schedtune`. The default value is 10ms.

Processes often do not use the full timeslice allocated. If, during execution, the thread becomes unrunnable (for example, it issues a synchronous I/O request), another thread is dispatched.

The process of switching from one task to another is known as context switching. When a context switch occurs, information must be saved to allow the process to be resumed again and the CPU instruction pipeline flushed. When a task is resumed, the saved information is retrieved and execution begins at the point where the context switch occurred.

When other processes have been dispatched in the meantime, or the process does not maintain an affinity with the CPU, additional work is required to restart a task:

- The Process and CPU cache are reloaded.
- Translation Lookaside Buffers (TLBs) may not contain the address translations for the process's virtual memory address space.
- The CPU pipeline is reloaded.
- The VMM may have paged-out memory required by the process.

High context switching results in reduced system throughput. A larger timeslice can reduce context switching. It can also make the system less responsive.

Example:

```
schedtune -t 2
```

Set the timeslice to 2 clock ticks (20ms). The default timeslice is one clock tick (10ms).

11.4 Managing Input/Output Resources

The disk subsystem is a critical I/O sub-system component. The configuration, file system, and file layout can make the difference between acceptable or unacceptable performance.

11.4.1 Monitoring I/O Using iostat

We used this command in 11.3.5, "Monitoring the CPU with iostat" on page 174 to determine the effect of I/O operations on CPU utilization.

Use iostat as a first step in investigating I/O performance problems. The first line on an iostat report is the I/O statistics since the system was started.

Example:

In Figure 58 on page 180, the disks hdisk9 and hdisk10 have the highest load. The disk hdisk9 has a 20 percent higher load than hdisk10.

The other drives share the remaining load evenly.

Moving load from `hdisk10` or `hdisk9` would provide a more balanced system and increase overall performance.

The high level of I/O wait on our system may also be reduced if the disks are more evenly balanced.

```
# iostat 1 1
tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          3.3        738.1              15.4      8.8       55.6      20.3

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk2           0.0       0.0       0.0       1052         0
hdisk0           5.0      39.7       6.0    1767844   2849263
hdisk1           4.9      47.4       5.8    1804877   3702047
hdisk3           3.8      17.3       3.0    1018922    997348
hdisk4           5.1      19.2       4.1     614302   1618864
hdisk5           6.8      35.7       6.5    1604006   2548204
hdisk6           8.4      31.5       6.7    1677430   1984584
hdisk7           0.0       0.0       0.0       2148         0
hdisk8           6.5      32.9       7.2    1287490   2542280
hdisk9          11.6     310.5      20.6   22447138  13663752
hdisk10          8.1     266.2      16.4   21835730   9119258
cd0              0.1       0.0       0.0        164         0
```

Figure 58. I/O Statistics since Boot Using iostat

11.4.2 Monitoring I/O Using lslv

This command displays the placement of logical volumes on physical partitions.

If the system has significant I/O dependencies, investigate the placement of files.

AIX storage allocation strategy divides a disk into 5 regions:

- Outer Edge
- Outer Middle
- Center
- Inner Middle
- Inner Edge

A logical volume can span several regions and can be placed across multiple disks. Avoid situations were logical volumes are fragmented across disk regions.

By spreading a logical volume across multiple disks, performance gains are achieved when:

- Several requests for files in the logical volume are issued.
- The files reside on different disks.

File placement should be carefully managed.

Example:

Figure 59 is an example of a logical volume that has been split into two blocks on a single physical disk.

```
# lslv -p hdisk1 lv11
hdisk1:lv11:/files/app
0035   0036   0037   0038   0039   0040   0041   0042   0043   0044      1-10
0045   0046   0047   0048   0049   0050   0051                           11-17

USED   USED   USED   USED   USED   USED   USED   USED   USED   USED     18-27
USED   USED   USED   USED   USED   USED   USED                           28-34

USED   USED   USED   USED   USED   USED   USED   USED   USED   USED     35-44
USED   USED   USED   USED   USED   USED   USED                           45-50

0001   0002   0003   0004   0005   0006   0007   0008   0009   0010     51-60
0011   0012   0013   0014   0015   0016   0017                           61-67

0018   0019   0020   0021   0022   0023   0024   0025   0026   0027     68-77
0028   0029   0030   0031   0032   0033   0034                           78-84
```

Figure 59. Logical Volume on Disk Sample

Analysis of this report:

The first part of the logical volume is in the inner region.

The last part is in the outer edge region.

A process using this volume would experience additional I/O waits when the disk heads move back and forth between these regions.

Conclusion: Access is not optimal.

11.4.3 Monitoring I/O Using fileplace

This command displays information about the placement of files in a logical volume. It should be used on files that are heavily accessed.

Note

This command does not display the file placement of a remote file (a Network File System). You must run the `fileplace` command directly on the file server.

A file spread across a logical volume is not efficient when the volume is split across disk regions. If the logical volume is spread across multiple disks, the file placement is important. A large file heavily accessed and spread across multiple disks is more efficient to access.

Example:

Figure 60 is an example of using `fileplace` to view the placement of a file in a logical volume.

```
# fileplace -piv /config/config.app

File: config.app  Size: 1015808 bytes  Vol: /dev/lv31
Blk Size: 4096  Frag Size: 4096  Nfrags:  248   Compress: no
Inode: 782961  Mode: -rw-rw----  Owner: appadmin Group: app

INDIRECT BLOCK: 254629

Physical Addresses (mirror copy 1)                                Logical Fragment
----------------------------------                                ----------------
0255291-0255302  hdisk3          12 frags    49152 Bytes,  4.8%  1071899-1071910
0255309-0255332  hdisk3          24 frags    98304 Bytes,  9.7%  1071917-1071940
0255338-0255365  hdisk3          28 frags   114688 Bytes, 11.3%  1071946-1071973
0255373-0255380  hdisk3           8 frags    32768 Bytes,  3.2%  1071981-1071988
0255386-0255453  hdisk3          68 frags   278528 Bytes, 27.4%  1071994-1072061
0255464-0255479  hdisk3          16 frags    65536 Bytes,  6.5%  1072072-1072087
0255485-0255520  hdisk3          36 frags   147456 Bytes, 14.5%  1072093-1072128
0255527-0255554  hdisk3          28 frags   114688 Bytes, 11.3%  1072135-1072162
0255562-0255589  hdisk3          28 frags   114688 Bytes, 11.3%  1072170-1072197

248 frags over space of 299 frags:  space efficiency = 82.9%
9 fragments out of 248 possible:  sequentiality = 96.8%
```

Figure 60. File Placement within a Logical Volume

To assess the performance effect of this file fragmentation, an understanding of how the file is used by the application is required:

- If the application is primarily accessing this file sequentially, the logical fragmentation is more important. At the end of each fragment, read ahead stops. The fragment size is therefore very important.

- If the application is accessing this file randomly, the physical fragmentation is more important. The closer the information is in the file, the less latency there is when accessing.

This file is efficient (but not optimal) when accessed sequentially or randomly. The fragments are large and close to each other.

11.4.4 Monitoring I/O Using filemon

This command reports I/O activity.

Detailed information on the disk activity and the VMM usage of the page area is provided. The report shows the most active segments (files) and the most active volumes (both physical and logical).

This information is used to evaluate whether the placement of files and logical volumes is appropriate. It can also be used to determine the optimal locations of files and logical volumes.

Example:

As an example, on a lightly loaded system, we copied a small file to /dev/null. During the copy, we used `filemon` to monitor the system I/O activity.

Figure 61 shows the procedure we used to produce the `filemon` report.

```
# filemon ; cp dummy_file /dev/null ; trcstop

Tue Sep 29 15:43:09 1998
System: AIX sp3en0 Node: 4 Machine: 000081007000

1.781 secs in measured interval
Cpu utilization:  6.1%
```

Figure 61. Using filemon

Note

When investigating I/O activity using `filemon`, monitor the I/O activity of the system without adding an additional dummy load.

11.4.4.1 Identification of the Most Active Files

The Most Active Segments report produced by `filemon` shows the most active segments. Use this report to determine files that should be separated.

In Figure 62, the most active segment was on logical volume `hd2`; the inode is `13`.

```
Most Active Segments
------------------------------------------------------------------------
  #MBs  #rpgs  #wpgs  segid  segtype             volume:inode
------------------------------------------------------------------------
   0.0      1      0   7ebd  persistent          /dev/hd2:13
```

Figure 62. Identifying the Most Active Segments Using filemon

To find the name of the file, translate the inode for the volume into a file name. Figure 63 on page 184 is an example of how to do this. On our system the most active file was /tmp/dummy_file. This is the file we copied to /dev/null.

```
# find `df | grep "/dev/hd2 " | cut -c58-132` -inum 13 -print >\
/tmp/dummy_file
```

Figure 63. Translating Volume and inode to File Name

We created script fn, shown in Figure 64, to make this process easier. To translate inode 13 on the logical volume using this script, type `fn -v /dev/hd2 13`.

```
#!/bin/ksh

HEAD=`df . | head -n1`
VOLUME=`df . | grep -v "${HEAD}" | cut -d' ' -f1`

while getopts v: FLAG
do
   case ${FLAG} in
   v) VOLUME="${OPTARG}";;
   ?) echo "Do not understand ${OPTARG}"
      exit 2;;
   esac
done

shift $((OPTIND -1))

FILEROOT=`df | grep -v "${HEAD}" | grep "${VOLUME} " | cut -c58-132`

if [ -z "${FILEROOT}" ]; then
   echo "${VOLUME} is not mounted."
   exit 2
fi

for INODE in $*
do
   echo ${INODE} | grep -qx "[0-9]*"
   if [ $? != 0 ]; then
      echo "${INODE} is not a valid inode"
   else
      FILENAME=`find ${FILEROOT} -inum ${INODE} -print`
      if [ -z "${FILENAME}" ]; then
         echo "Inode ${INODE} on logical volume ${VOLUME} was not found"
      else
         echo "Inode ${INDOE} on logical volume ${VOLUME} is ${FILENAME}"
      fi
   fi
done
```

Figure 64. Sample Script to Obtain File Name

Script fn has the single option -v. This specifies the logical volume, for example, fn -v /dev/hd2. If the logical volume is not specified, the logical volume for your current working directory is used.

Any number of inodes can be specified. Use a space between each inode address, for example, fn -v /dev/hd2 13 18 27. This example will translate inodes 13, 18, and 27 into their respective file names on the logical volume hd2.

11.4.4.2 Identification of the Most Active Logical Volume

The Most Active Logical Volumes report produced by `filemon` shows the most active logical volumes.

In our example, shown in Figure 65, the most active logical volume is also the most active physical volume.

```
Most Active Logical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume              description
------------------------------------------------------------------------
  0.01      8      0    2.2  /dev/hd2            /tmp
```

Figure 65. Identifying the Most Active Logical Volume Using filemon

Values to check:

`util`: utilization as a percentage
`KB/s`: average number of kilobytes transferred per second

A high value for either of these two columns indicates a bottleneck.

11.4.4.3 Identification of the Most Active Physical Volume

The Most Active Physical Volumes report produced by filemon shows the physical volumes that are most active. Often, the most active physical volume is the one containing the most active file or logical volume.

In Figure 66, `/dev/hdisk2` is the most active physical volume.

```
Most Active Physical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume              description
------------------------------------------------------------------------
  0.01      8      0    2.2  /dev/hdisk2         2.0 GB SCSI Disk Drive
```

Figure 66. Identifying the Most Active Physical Volume Using filemon

Values to check:

`util`: utilization as a percentage
`KB/s`: average number of kilobytes transferred per second

A high value for either of these two columns indicates a bottle neck.

11.4.4.4 Assessment of VMM on the I/O Resources

The performance of the I/O subsystems plays a critical role in system throughput.

How quickly and efficiently an application can access a file and how efficiently the VMM can manage the application's memory requirements is a significant limiting factor in system throughput.

The Detailed VM Segment Stats report produced by filemon is used to assess the I/O activity generated by the VMM.

Figure 67 is an example of this report.

```
------------------------------------------------------------------------
Detailed VM Segment Stats   (4096 byte pages)
------------------------------------------------------------------------

SEGMENT: 7ebd  segtype: persistent   volume: /dev/hd2  inode: 25
segment flags:pers
reads:1(0 errs)
  read times (msec):avg  19.761 min  19.761 max  19.761 sdev   0.000
  read sequences: 1
  read seq. lengths:avg     1.0 min       1 max       1 sdev     0.0
```

Figure 67. Investigating the I/O Performance Effect on VMM

Analysis of this report:

reads: The number of reads (indicates the number of page requests).

read sequences: The number of sequential page reads (indicates the level of fragmentation in memory).

avg: The length of time on average that I/O operations took (indicates the page space efficiency).

sdev: Access times with significant variation (indicates the level of disk competition)

11.4.4.5 Assessment of I/O Utilization in a Logical Volume

In 11.4.4.2, "Identification of the Most Active Logical Volume" on page 185, the filemon report Most Active Logical Volumes was discussed. That report was a summary of the I/O activity on logical volumes.

The filemon report Detailed Logical Volume Stats provides a breakdown of the I/O activity in logical volumes.

Figure 68 shows an example of this report.

```
------------------------------------------------------------------------
Detailed Logical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------

VOLUME: /dev/hd2  description: /usr
reads:1(0 errs)
  read sizes (blks): avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):avg  19.729 min  19.729 max  19.729 sdev   0.000
  read sequences: 1
  read seq. lengths:avg     8.0 min       8 max       8 sdev     0.0
seeks:1(100.0%)
  seek dist (blks):init 398944
time to next req(msec): avg 1737.330 min 1737.330 max 1737.330 sdev   0.000
throughput:2.2 KB/sec
utilization:0.01
```

Figure 68. Investigating the I/O Performance of Logical Volumes

Analysis of this report:

`utilization`: A low utilization indicates free capacity for higher volumes of data transfers.

`avg`: The average access time is an indication of efficiency.

`sdev`: Variation in access times indicates the level of fragmentation.

11.4.4.6 Assessment of I/O Utilization within a Physical Volume

In 11.4.4.3, “Identification of the Most Active Physical Volume” on page 186, the *filemon* report Most Active Physical Volumes was discussed. This report displayed a summary of the I/O activity for physical volumes.

The Detailed Physical Volume Stats report that *filemon* produces provides a breakdown of the I/O activity on the physical volumes.

Figure 69 on page 189 is an example of the Detailed Physical Volume Stats *filemon* report.

```
------------------------------------------------------------------------
Detailed Physical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------

VOLUME: /dev/hdisk2  description: 2.0 GB SCSI Disk Drive
reads:1(0 errs)
  read sizes (blks): avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):avg  19.623 min  19.623 max  19.623 sdev   0.000
  read sequences: 1
  read seq. lengths:avg     8.0 min       8 max       8 sdev     0.0
seeks:1(100.0%)
  seek dist (blks):init 2271072
  seek dist (%tot blks):init 57.74342
time to next req(msec): avg 1737.371 min 1737.371 max 1737.371 sdev   0.000
throughput:2.2 KB/sec
utilization:0.01
```

Figure 69. Investigating the I/O Performance of Physical Volumes

Analysis of this report:

`utilization`: A low utilization indicates free capacity for higher volumes of data transfer.

`avg`: The average access time is an indication of efficiency.

`sdev`: Variation in access times indicates the level of fragmentation.

11.4.5 Managing Fragmentation

The following sections describe how to manage fragmentation.

11.4.5.1 Managing File System Fragmentation Using defragfs

The command `defragfs` is used to reduce file system fragmentation.

File system fragmentation is reduced by reorganizing the file allocations contiguously. It occurs when disk blocks are allocated and deallocated by applications.

To check if a file system is fragmented and would benefit from being defragmented (a process that, on large partitions, will take considerable time and resources), use `defragfs -q`.

Example:

Figure 70 on page 190 is a sample `defragfs` report showing the level of fragmentation in a file system.

```
# defragfs -q /dev/hd2
statistics before running defragfs:
number of free fragments 116123
number of allocated fragments 14949
number of free spaces shorter than a block 1
number of free fragments in short free spaces 2
```

Figure 70. Querying State of Fragmentation within a File System

Check the following values:

- `Number of free spaces shorter than a block`
- `Number of free fragments`

If the `number of free spaces shorter than a block` is high or close to the `number of free fragments`, use `defragfs` to consolidate the free space.

Example:

```
defragfs /dev/hd2
```

This will defragment and consolidate the free space on logical volume `hd2`.

11.4.5.2 Managing Volume Group Fragmentation

The command `reorgvg` is used to move the placement of logical volumes in physical volumes.

If a logical volume is fragmented across a physical volume (refer to 11.4.2, “Monitoring I/O Using lslv” on page 180 for information on checking the placement of logical volumes), reorganize the logical volume.

Example:

```
reorgvg appvg lv11 lv12
```

Reorganize the placement of logical volumes `lv11` and `lv12` in the volume group `appvg`.

11.4.5.3 Rebuilding a File System
Rebuild file systems that are heavily fragmented to increase their efficiency.

Scenario:

The file system /appdata is heavily fragmented. This file system is contained on logical volume lv31.

The following steps are performed to rebuild the file system:

1. Back up the file system.
2. Verify the backup.
3. Unmount the file system; in our case, use unmount /appdata.
4. Remake the file system using mkfs; in our case, use mkfs /dev/lv31. This is destructive, and you are asked to verify that you actually want to do this.
5. Remount the file system; in our case, use mount /dev/lv31 /appdata.
6. Restore the backup.

11.4.6 Tuning Kernel I/O Parameters
Kernel parameters can be changed to improve I/O throughput. Before tuning any kernel parameters, the workload I/O requirements must be determined.

Tuning kernel I/O parameters is a process of trading off the efficiency of one form of access off against that of another.

Place changes made with vmtune into /tftpboot/tuning.cust. This ensures that, when the system is rebooted, the changes will remain in effect.

Note

In the examples given, we have set the system parameters to the recommended default values for our installation of AIX 4.3.2.0.

11.4.6.1 Tuning Sequential Read Ahead Using vmtune
If the VMM determines that sequential access to a file is being made, it schedules additional reads (read ahead). Read ahead is used to retrieve file blocks before they are required.

AIX has two kernel parameters that the VMM uses to determine when read ahead should happen and how many blocks are to be read ahead.

The kernel parameter minpgahead sets the number of sequential file blocks that must be sequentially read by an application before read ahead will begin.

This parameter also sets the number of blocks that are to be read ahead initially.

The kernel parameter maxpgahead sets the maximum number of file blocks that are read ahead on behalf of the application.

Table 26 shows how read ahead works when minpgahead is 2 and maxpgahead is 8.

Table 26. Tuning Read Ahead

Step	File Access	Read Ahead
1	1st file block	No blocks read ahead
2	2nd file block	Blocks 3 - 4 read ahead
3	4th file block	Blocks 5 - 8 read
4	8th file block	Blocks 9 - 16 read
5	16th file block	Blocks 17 - 24 read
6	Out-of-sequence file block	Read ahead canceled

The processes shown in Table 26 work as follows:

1. An application accesses the first file block.
2. The next file block is read by the application (file block 2). The number of file blocks accessed in sequence equals minpgahead. Two additional blocks are read ahead.
3. The application continues to read file blocks sequentially. File block 3 is the last file block read ahead. When accessed, four additional file blocks are read ahead.
4. File blocks continue to be accessed sequentially. File block 8 is the last file block read ahead. When accessed, eight additional file blocks are read ahead. Eight is the maximum number to read ahead (value of maxpgahead).
5. As the file access continues sequentially, accessing the last block read ahead schedules an additional eight blocks for read ahead.
6. The file is accessed out of sequence. Read ahead is canceled.

Note

- The values of minpgahead and maxpgahead should always be powers of two, and the maxpgahead value must be greater than or equal to the minpgahead value.
- If the minpgahead value is set to 0, sequential read ahead will not be instigated by the VMM.

Example:

```
vmtune -r 2 -R 8
```

Set minpgreadahead to **2**, and maxpgahead to **8**. Read ahead begins when a second file block is accessed sequentially. A maximum of eight file blocks at a time are read ahead.

11.4.6.2 Using I/O Pacing

Some processes generate output very quickly and in large quantities. This can cause large backlogs in the number of pending I/O operations.

A large backlog of work in the I/O subsystem will delay processes that have to wait on an I/O operation.

I/O pacing is an option that will limit the number of pending I/O requests against a file or segment. When the number of pending writes for a file exceeds this point, the process is suspended until the level of outstanding I/O requests is reduced.

I/O pacing penalizes processes that rapidly issue I/O write requests in favor of other less I/O-demanding processes.

The kernel parameters maxpout and minpout configure I/O pacing:

maxpout	Sets the maximum number of I/O write requests that are allowed against a single file.
minpout	Sets the number of I/O requests against a single file before the issuing processes are resumed.

Note

Setting minpout and maxpout to 0 disables I/O pacing.

Example:

```
chdev -l sys0 -a maxpout='33' -a minpout='16'
```

Set the maximum number of pending I/O write requests outstanding against a file to 33 before suspending the process.

Once the process is suspended, it will not be resumed until the outstanding I/O write requests against the file drop to 16.

11.4.6.3 Tuning Asynchronous Disk I/O

The AIX kernel uses a kernel process (kproc) to manage Asynchronous I/O. Every Asynchronous I/O operation requires a kproc to manage the request. The number of kprocs determines the number of Asynchronous I/O operations that can be performed together.

If there are two kprocs, only two Asynchronous I/O operations can be performed together.

The number of kproc servers is set with the kernel parameters minservers and maxservers:

minservers:	The number of kprocs to start when the system is booted.
maxservers:	The maximum number of kprocs the kernel can have running at any one point in time. This limits the maximum number of asynchronous I/O operations.

Example:

```
chdev -l aio0 -a minservers='10' -a maxservers='20'
```

This sets the minimum number of kprocs to 10, and the maximum number of kprocs to 20.

Note

For applications with high Asynchronous I/O requirements, set maxservers to 10 times the number of drives, and set minservers to half this value.

11.5 Managing Network Resources

The network in an RS/6000 SP forms the complex. Without it, the RS/6000 SP would be a set of individual nodes operating independently.

Network communication is a combination of hardware and complex software.

It is important that network parameters be set appropriately. The switch, for example, is capable of shifting 120+ MB of data per second. With incorrect configuration, the switch throughput may only be a few hundred kilobytes.

The importance of network tuning depends on how the RS/6000 SP complex is configured:

- If the RS/6000 SP complex is used as a convenient way to administer a large number of disparate computer systems, network tuning is of limited value.
- If there is tight coupling between nodes, and the work load is divided across the nodes, the network is a critical component.

  The network is one of the slowest components in a complex. In any RS/6000 SP complex with tight coupling between the nodes, investigation of the network is a high priority.

11.5.1 Monitoring the Network Using Adapter Statistics

Most network adapters have `stat` commands which list the device driver statistics.

The following `stat` commands can be used with each of the specified adapters to obtain the device driver statistics:

`entstat`: Ethernet adapter (ent0..n)

`tokstat`: Token ring adapter (tr0..n)

`fddistat`: Fiberoptic Distributed Data Interface (FDDI) adapter (fddi0..n)

`atmstat`: Asynchronous Transfer Mode (ATM) adapter (atm0..n)

`estat`: RS/6000 SP switch (css0..n)

An estimate of adapter utilization is calculated from the bytes sent and received.

Using `estat css0` on our system (shown in Figure 71 on page 196) we can calculate that 0.8 MB per hour of bandwidth is used: (bytes sent + bytes received) / elapsed time.

The S/W Transmit Queue Overflow value should also be checked. Transmit queue overflows (flooding) cause delays in network traffic on the adapter.

```
# /usr/lpp/ssp/css/estat css0
-------------------------------------------------------------
CSS STATISTICS (css0) :
Elapsed Time: 35 days 5 hours 5 minutes 25 seconds

Transmit Statistics:                          Receive Statistics:
--------------------                          -------------------
Packets: 3352220                              Packets: 3341258
Bytes: 347889708                              Bytes: 395510912
Interrupts: 0                                 Interrupts: 3070419
Transmit Errors: 0                            Receive Errors: 0
Packets Dropped: 0                            Packets Dropped: 0
Max Packets on S/W Transmit Queue: 0          Bad Packets: 0
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 0

Broadcast Packets: 0                          Broadcast Packets: 0

General Statistics:
-------------------
No mbuf Errors: 0
```

Figure 71. Network Adapter Statistics

The adapter statistics are reset using the `estat -r` option. For example, `estat -r css0` resets the device driver statistics for the network adapter `css0`.

Determining utilization using device driver statistics requires knowledge of each adapter's rated bandwidth, the protocol used, and how it is used.

Examples:

- In most environments Ethernet adapters driven above 25 percent to 30 percent will result in serious degradation of the network link. Degradation of the network link is often seen with 10 percent loading.

  A good indicator of an overloaded network is:
  Collisions / Transmit Packets > 0.1
- If you have an application that transfers large volumes of data on a dedicated link between two hosts in a single direction, the transfer rates could exceed 80 percent of the theoretical maximum transfer when tuned correctly.

- Using ATM, 50 percent to 60 percent of the theoretical transfer rate can be achieved on an Ethernet adapter before serious degradation of the network is seen.
- A token ring will achieve 60 percent to 70 percent of the theoretical transfer rate before the network link becomes significantly degraded.
- In lab testing of the RS/6000 SP Switch, transfer rates exceeding 120 MB per second have been achieved. The Global Parallel File System (GPFS) for the RS/6000 SP can get very close to these transfer rates when the switch is dedicated to GPFS use and the link is tuned correctly.

11.5.2 Monitoring the Switch with vdidl2 or vdidl3

The commands vdidl2 and vdidl3 provide switch memory pool statistics.

Which command to use depends on the type of switch. For the HiPS switch, use vdidl2, and, on the SP Switch, use vdidl3.

Figure 72 on page 198 is an example of the report produced using vdidl3.

```
# /usr/lpp/ssp/css/vdidl3 -i
get ifbp info...

send pool: size=2097152 anchor@=0x50002e00 start@=0x51210000
tags@=0x50004000
bkt   allocd     free  success     fail    split     comb    freed
 12        0        0    12866        0    25655        0        0
 13        0        0        3        0       12        0        0
 14        0        0    25683        0    25684        0        0
 15        0        0        0        0        0        0        0
 16        0       32       47        0        0        0        0

rsvd pool: size=262144 anchor@=0x510d4200 start@=0x51410000
tags@=0x50003d00
bkt   allocd     free  success     fail    split     comb    freed
 12        0        0        0        0        0        0        0
 13        0        0        0        0        0        0        0
 14        0        0        0        0        0        0        0
 15        0        0        0        0        0        0        0
 16        0        4        0        0        0        0        0

recv pool: size=2097152 anchor@=0x50002000 start@=0x51450000
tags@=0x510d3400
bkt   allocd     free  success     fail    split     comb    freed
 12        0        0        0        0        0        0        0
 13        0        0        0        0        0        0        0
 14        0        0        0        0        0        0        0
 15        0        0        0        0        0        0        0
 16        0        0        0        0        0        0        0
```

Figure 72. Switch Pool Statistics Using vdidl3

Interpretation of the report:

The values that should be considered are the `success`, `fail`, and `split` columns. Interpretation of these values is as follows:

- Allocation of switch pool memory is in request blocks. A block size is always 2^`blk`. For example, in the row were column `blk` is `12`, the block allocation size is 2^`12`=4K.
- The column `success` shows the number of successful attempts to allocate a requested memory block.

- The column `fail` shows the number of failed attempts to allocate a requested contiguous memory block.
- The column `split` shows the number of times a block could not be allocated contiguously in the memory pool and was allocated as fragmented memory blocks.
- The number of memory block allocation failures is fails - split.

As the switch begins to reach the transfer limit, the switch pools become fragmented, and the switch begins to fail requests for memory block allocations.

The SP switch receive pool (recv pool) statistics are no loner updated. The receive pool is now implemented in the switch using microcode on the adapter.

11.5.3 Monitoring the Network with netstat

This command is used to assess the network load and the reliability of the network. It is traditionally used for problem determination. We have found this command to be useful for determining the network load, and whether the network is congested.

11.5.3.1 Determine the Proportion of Network Traffic for an Adapter

Use `netstat -I` to compare the network traffic on a selected adapter with the total volume of network traffic.

Figure 73 is an example of a `netstat -I` report. This report was produced while a large file was transferred using ftp (a file transfer application) between two nodes. The high-speed switch css0 was chosen to do this transfer.

```
# netstat -I css0 1
    input   (css0)     output           input   (Total)    output
 packets  errs  packets  errs colls  packets  errs  packets  errs colls
  125696     0   110803     0     0   356878     0   287880     0     0
     119     0      216     0     0      123     0      221     0     0
     117     0      222     0     0      120     0      224     0     0
     115     0      225     0     0      117     0      227     0     0
     115     0      202     0     0      117     0      204     0     0
     115     0      207     0     0      117     0      209     0     0
     116     0      201     0     0      118     0      203     0     0
     115     0      211     0     0      118     0      213     0     0
```

Figure 73. Viewing the Network Load Using netstat

The report is divided into five columns for the adapter, and five columns for the total network utilization:

1. Incoming packets.
2. Number of error packets received. An error causes a retransmission of the packet.
3. Number of packets sent.
4. Number of error packets sent (the number of packets which were re-requested because of a transmission error).
5. Number of collisions (colls). When information is sent using a network adapter, a component of Transmission Control Protocol (TCP) is a collision detection algorithm. The algorithm works as follows:
    1. Before sending, check that the network connection is not in use.
    2. When the network connection is free, begin writing the packet onto the network.
    3. Check that in the very short amount of time between determining that the network connection was free and beginning to write the packet, no other network adapter started to write a packet. When this happens, it is called a network collision.
    4. The adapters abort the write operation.
    5. The adapters wait for a random period of time before trying again.

    Collisions affect network throughput: the adapters involved are forced to spend time waiting, and network bandwidth is wasted when the collision occurs.

    A few collisions will happen in a network from time to time and are acceptable. If a lot of collisions are detected, it indicates the network is overloaded.

Note

- Adapters involved in a network collision can be in the same host or in different hosts.
- Packet numbers do not directly translate into network utilization, because packet sizes vary. Packet numbers are only a guide to network load.

11.5.3.2 Determining an Adapter’s Packet Size

Use netstat -i to find the Maximum Transmission Unit (MTU) size of each network adapter and the number of packets each adapter has received or sent.

In Figure 74 on page 201, the adapter css0 has an MTU size of 65520 bytes. Figure 73 on page 199 shows that 115 packets were received and 205 packets were sent on average. If each packet fully utilized the MTU, a transfer rate of 20 MB per second would have been achieved. In our example, using ftp, we averaged under 4 MB per second.

Using a single channel of the switch, with optimal packaging, the SP Switch we were using can transfer 30 MB per second.

Note

Be careful when interpreting figures which are not rigidly and precisely defined. Packets are good examples of values that can only be interpreted in conjunction with other values.

```
# netstat -i
Name  Mtu   Network     Address           Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 link#1                          15509     0    15519     0     0
lo0   16896 127         loopback            15509     0    15519     0     0
lo0   16896 ::1                             15509     0    15519     0     0
en0   1500  link#2      2.60.8c.e8.fc.b3    19362     0    15659     0     0
en0   1500  192.168.5   sp5n13.msc.itso.i   19362     0    15659     0     0
css0  65520 link#3                          57230     0    85174     0     0
css0  65520 192.168.15  sp5sw13.msc.itso.   57230     0    85174     0     0
```

Figure 74. Using netstat to Find MTU Size

11.5.3.3 Network Memory Buffer Allocation Statistics

If memory buffer allocation requests fail, the request is lost and more memory needs to be allocated to the network memory pool. For an explanation on how to do this, refer to 11.5.7.1, “Setting Upper Memory Bound for Communications” on page 216.

Figure 75 on page 202 is an example of using `netstat -m` to check the network memory buffer allocation statistics. Check the column *failed* to ensure that the communication subsystem has sufficient memory allocated.

```
# netstat -m

Kernel malloc statistics:

******* CPU 0 *******
By size       inuse     calls failed    free   hiwat   freed
32              269    150117      0     115     640       0
64              159      1367      0      33     320       0
128             108      2213      0     148     160       5
256             260   6736146      0     380     384      13
512             186    614975      0      38      40       5
1024             68     98827      0      68     100       0
2048              0    132368      0      34     100       0
4096             19     33252      0      37     120       0
8192              0      1495      0       0      10       0
16384             1     25004      0      20      24     145
32768             1         1      0       0     512       0

By type       inuse     calls failed  memuse  memmax  mapb

Streams mblk statistic failures:
0 high priority mblk failures
0 medium priority mblk failures
0 low priority mblk failures
```

Figure 75. Network Memory Buffer Allocation Statistics

11.5.3.4 Network Connections

Use `netstat -d` to determine:

- The number of network connections
- The network protocol used for each connection
- The application using the network connection
- The process assigned the network connection
- The host or client with which the network connection has been made

Figure 76 on page 203 shows an example of using `netstat -d` to list the currently open network connections (we reduced the size of this list). In this example, most network connections are established with `TOT39`.

```
# netstat -d
Active Internet connections
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0  sp4cw0.msc.itso..33664 TOT39.itso.ibm.c.6000  ESTABLISHED
tcp4       0      0  sp4cw0.msc.itso..telne TOT104.itso.ibm..3529  ESTABLISHED
tcp4       0      0  sp4cw0.msc.itso..33635 TOT39.itso.ibm.c.6000  ESTABLISHED
tcp4       0      0  sp4en0.msc.itso..sdr   sp4en0.msc.itso..33285 ESTABLISHED
tcp4       0      0  sp4en0.msc.itso..33285 sp4en0.msc.itso..sdr   ESTABLISHED
tcp4       0      0  sp4en0.msc.itso..sdr   *.*                    LISTEN
tcp4       0      0  sp4cw0.msc.itso..33102 TOT39.itso.ibm.c.6000  ESTABLISHED
tcp4       0      0  sp4cw0.msc.itso..33098 TOT39.itso.ibm.c.6000  ESTABLISHED
tcp4       0      0  sp4cw0.msc.itso..33097 TOT39.itso.ibm.c.6000  ESTABLISHED
tcp4       0      0  sp4cw0.msc.itso..33095 TOT39.itso.ibm.c.6000  ESTABLISHED
tcp4       0      0  sp4cw0.msc.itso..33090 TOT39.itso.ibm.c.6000  ESTABLISHED
tcp4       0      0  sp4en0.msc.itso..hardm sp4en0.msc.itso..33085 ESTABLISHED
tcp4       0      0  sp4en0.msc.itso..33085 sp4en0.msc.itso..hardm ESTABLISHED
tcp4       0      0  sp4en0.msc.itso..17661 *.*                    LISTEN
tcp4       0      0  sp4en0.msc.itso..33064 sp4en0.msc.itso..sdr   CLOSE_WAIT
tcp4       0      0  sp4en0.msc.itso..haemd *.*                    LISTEN
tcp4       0      0  sp4en0.msc.itso..hardm sp4en0.msc.itso..32837 ESTABLISHED
tcp4       0      0  sp4en0.msc.itso..32837 sp4en0.msc.itso..hardm ESTABLISHED
```

Figure 76. Network Connections

11.5.4 Monitoring Network Traffic Using iptrace

This command monitors all network traffic on the host system. It is used to identify network problems. This is not a tool that will be used routinely. `iptrace` degrades system performance because it collects substantial amounts of data for analysis. Each packet sent or received is captured.

Figure 77 is an example of the procedure to run `ipctrace`. To illustrate the use of `iptrace` we used it on an unloaded node, and used `echo` to write a line of text to a Network File System (NFS). We then stopped `ipctrace` and formatted the report using `ipreport`.

```
# startsrc -s iptrace -a "-i css0 /tmp/iptrace_log"
0513-059 The iptrace Subsystem has been started. Subsystem PID is 12984.
# echo " Network Traffic \n" >> /network_drive/test_data
# stopsrc -s iptrace
0513-044 The stop of the iptrace Subsystem was completed successfully.
# ipreport /tmp/ipctrace_log > /tmp/iptrace_report
# pg /tmp/iptrace_report
```

Figure 77. Running iptrace

The report produced contained a number of transmissions; both send and receive network operations were captured.

The first packet of information reported by `ipctrace` is shown in Figure 78. From this report, we can determine that the packet was destined for sp5sw05 and contained 188 bytes.

```
IPTRACE version: 2.0

TRACING DROPPED 1457 PACKETS after packet 0. TOTAL DROPPED THIS TRACE=1457

====( 188 bytes transmitted on interface css0 )==== 14:21:39.320966239
OTHER packet   (IP)
IP header breakdown:
        < SRC =   192.168.15.13 >  (sp5sw13.msc.itso.ibm.com)
        < DST =    192.168.15.5 >  (sp5sw05.msc.itso.ibm.com)
        ip_v=4, ip_hl=20, ip_tos=0, ip_len=188, ip_id=19439, ip_off=0
        ip_ttl=60, ip_sum=92ea, ip_p = 6 (TCP)
TCP header breakdown:
        <source port=33023, destination port=2049(shilp) >
        th_seq=b34e1e2c, th_ack=e4ee7e09
        th_off=8, flags<PUSH | ACK>
        th_win=38886, th_sum=6907, th_urp=0
00000000     0101080a 36153fd8 36154042 80000084     |....6.?.6.@B....|
00000010     1525f276 00000000 00000002 000186a3     |.%.v............|
00000020     00000003 00000001 00000001 00000038     |...............8|
00000030     361519b3 00000006 7370356e 31330f0d     |6.......sp5n13..|
00000040     00000000 00000000 00000007 00000000     |................|
00000050     00000002 00000003 00000007 00000008     |................|
00000060     0000000a 0000000b 00000000 00000000     |................|
00000070     00000020 000a000a 00000003 000a0000     |... ............|
00000080     00143614 df980000 000a0000 00023613     |..6...........6.|
00000090     97e10000                                |....            |
```

Figure 78. First Transmission Block Captured Using iptrace

Figure 79 on page 205 shows the return packet received from sp5n05 in response.

```
====( 168 bytes received on interface css0 )==== 14:21:39.321888532
OTHER packet   (IP)
IP header breakdown:
        < SRC =    192.168.15.5 >  (sp5sw05.msc.itso.ibm.com)
        < DST =   192.168.15.13 >  (sp5sw13.msc.itso.ibm.com)
        ip_v=4, ip_hl=20, ip_tos=0, ip_len=168, ip_id=27499, ip_off=0
        ip_ttl=60, ip_sum=7382, ip_p = 6 (TCP)
TCP header breakdown:
                             <source port=2049(shilp), destination port=33023 >
        th_seq=e4ee7e09, th_ack=b34e1eb4
        th_off=8, flags<PUSH | ACK>
        th_win=30016, th_sum=b09e, th_urp=0
00000000     0101080a 36154042 36153fd8 80000070     |....6.@B6.?....p|
00000010     1525f276 00000001 00000000 00000000     |.%.v............|
00000020     00000000 00000000 00000000 00000001     |................|
00000030     000001ff 00000001 00000000 00000003     |................|
00000040     00000000 0000009a 00000000 00001000     |................|
00000050     00000000 0000000c 00000000 000a000a     |................|
00000060     00000000 00000014 3615188a 255fdb09     |........6...%_..|
00000070     3614e394 025d5ad6 3614e484 0ab780aa     |6....]Z.6.......|
```

Figure 79. First Received Block Captured Using iptrace

This tool is very useful when troubleshooting a network problem because both sides of the network conversation can be seen.

The `ipctrace` command is used to verify network tuning. For example, after increasing the MTU size, it is possible to determine whether the applications took advantage of this change by looking at the network transmissions.

11.5.5 Monitoring the Network Using netpmon

This command monitors and then reports the network activity. It provides reports that are used to determine the level of network utilization.

Several reports summarize the network activity. For each summary report, a detailed report is also produced.

This command utilizes the system `trace` facility to obtain the network statistics. Running this command consumes system resources and generates a large volume of information in memory. We do not recommend running this program routinely or for long periods of time.

We recommend that this command be used to take a snapshot of the network during a short period of time when there is a network performance problem.

Example:

Using an unloaded node, we monitored the network traffic with `netpmon`. Figure 80 shows the procedure used.

```
# netpmon -o /tmp/netmon_report

Enter the "trcstop" command to complete netpmon processing

# trcstop
[netpmon: Reporting started]

[netpmon: Reporting completed]

[netpmon: 112.168 secs in measured interval]
```

Figure 80. Using netpmon

In our example, one node is an NFS server (sp5n05), and one node is an NFS client (sp5n13).

The sample `netpmon` reports which follow are a mix taken from these nodes. When presenting a sample `netpmon` report of network traffic, the report is from the client node. When presenting a sample `netpmon` report of NFS activity, the report is from the server node.

11.5.5.1 Identification of Network CPU Utilization by Processes

The Process CPU Usage Statistics report produced by `netpmon` shows the effect processes using the network services have on CPU utilization.

Figure 81 on page 207 is an example of the Process CPU Usage Statistics report, which displays the 20 most active processes utilizing the CPU. The report also displays the CPU utilization for network services for each of these processes.

Applications with high CPU utilization for network services are applications network tuning should be biased toward.

```
Process CPU Usage Statistics:
-----------------------------
                                                   Network
Process (top 20)             PID  CPU Time   CPU %   CPU %
----------------------------------------------------------
kbiod                       7756   11.4033   0.765   0.708
ftp                        13634    9.4786   0.636   0.240
netpmon                    10236    4.5108   0.303   0.000
cat                        13628    4.2828   0.287   0.001
gil                         3096    3.6692   0.246   0.246
cp                         12836    3.0610   0.205   0.000
trace                      15048    1.8723   0.126   0.000
hatsd                      17602    1.5320   0.103   0.000
syncd                       4936    0.5629   0.038   0.000
ksh                        12226    0.1209   0.008   0.000
ls                         13630    0.1090   0.007   0.000
netm                        2838    0.0864   0.006   0.006
swapper                        0    0.0845   0.006   0.000
xntpd                      10330    0.0564   0.004   0.000
ping                       13632    0.0350   0.002   0.000
harmld                     17398    0.0257   0.002   0.000
ls                         13626    0.0208   0.001   0.000
hagsd                      16258    0.0189   0.001   0.000
trcstop                    13636    0.0175   0.001   0.000
init                           1    0.0109   0.001   0.000
----------------------------------------------------------
Total (all processes)              40.9961   2.750   1.201
Idle time                        1431.5116  96.027
```

Figure 81. CPU Network Utilization by Processes

In this example, NFS was the highest user of the network services (`kbiod` - Kernel Block Input/Output Daemon).

11.5.5.2 Identification of Network CPU Utilization by Interrupts

When the CPU is working and a device request or exception occurs, an interrupt is generated. The CPU saves the current task information and then vectors to a First Level Interrupt Handler (FLIH).

The FLIH routes the interrupt to a Second Level Interrupt Handler (SLIH).

The First Level Interrupt Handler CPU Usage Statistics report that `netpmon` produces shows the amount of CPU time spent in the FLIHs.

Figure 82 on page 208 shows an example of this report.

```
First Level Interrupt Handler CPU Usage Statistics:
---------------------------------------------------
                                                  Network
FLIH                              CPU Time   CPU %   CPU %
----------------------------------------------------------
PPC decrementer                     6.9969   0.469   0.000
external device                     2.2347   0.150   0.039
data page fault                     1.8917   0.127   0.000
UNKNOWN                             0.1432   0.010   0.000
instruction page fault              0.0006   0.000   0.000
program check                       0.0000   0.000   0.000
----------------------------------------------------------
Total (all FLIHs)                  11.2672   0.756   0.039
```

Figure 82. CPU Network Utilization by FLIH

The FLIHs are very fast and consume very little CPU time. Most of the interrupt handling is performed in the SLIH.

The Second Level Interrupt Handler CPU Usage Statistics report that netpmon produces shows the amount of CPU time that is spent in the SLIHs.

This report is shown in Figure 83.

```
Second Level Interrupt Handler CPU Usage Statistics:
----------------------------------------------------
                                                  Network
SLIH                              CPU Time   CPU %   CPU %
----------------------------------------------------------
entdd                               7.9074   0.530   0.530
cssdd3                              0.0798   0.005   0.000
ascsiddpin                          0.0256   0.002   0.000
ssapin                              0.0044   0.000   0.000
----------------------------------------------------------
Total (all SLIHs)                   8.0172   0.538   0.530
```

Figure 83. CPU Network Utilization by SLIH Summary Report

In our examples, the network interrupt CPU time was consumed by the FLIH servicing the `external device` (Figure 82) interrupt.

We used the Second Level Interrupt Handler CPU Usage Statistics report to determine which device/adapter generated the interrupts. In our example it was the device driver `entdd` (the Ethernet adapter device driver). Less than 0.6 percent of the CPU was used by this device driver.

The Detailed Second Level Interrupt Handler CPU Usage Statistics report produced by netpmon displays CPU service time statistics.

Use this report to assess the CPU time of each network interrupt. Use this assessment to calculate the effect of additional network traffic.

Figure 84 shows an example of the Detailed Second Level Interrupt Handler CPU Usage Statistics report. It shows that interrupts by the adapter ent0 (entdd) cost on average (avg) `0.018` milliseconds of CPU time.

```
Detailed Second Level Interrupt Handler CPU Usage Statistics:
-------------------------------------------------------------

SLIH: entdd
count:                  27194
  cpu time (msec):      avg 0.291   min 0.018   max 2.088   sdev 0.109

SLIH: cssdd3
count:                  2150
  cpu time (msec):      avg 0.037   min 0.021   max 0.393   sdev 0.016

SLIH: ascsiddpin
count:                  390
  cpu time (msec):      avg 0.066   min 0.028   max 0.183   sdev 0.032

SLIH: ssapin
count:                  76
  cpu time (msec):      avg 0.058   min 0.034   max 0.134   sdev 0.019

COMBINED (All SLIHs)
count:                  29810
  cpu time (msec):      avg 0.269   min 0.018   max 2.088   sdev 0.126
```

Figure 84. CPU Network Utilization by SLIH Detailed Report

11.5.5.3 Identification of Network Adapter Utilization

The Network Device-Driver Statistics (by Device) report produced by netpmon displays data throughput of each network adapter during the time monitored.

Figure 85 on page 210 shows an example of this report. The only adapter used was ent0.

```
Network Device-Driver Statistics (by Device):
---------------------------------------------
                        ----------- Xmit -----------   -------- Recv ---------
Device                   Pkts/s  Bytes/s  Util  QLen   Pkts/s  Bytes/s   Demux
------------------------------------------------------------------------------
ethernet 0                91.16   135284  0.0%108.712    50.18     3434  0.8160
```

Figure 85. Network Adapter Utilization Summary Report

In this example:

- Data throughput is 135 KB per second (`Xmit Bytes/s + Recv Bytes/s`).
- Average packet size is 981 bytes ((`Xmit pkts/s + Recv pkts/s`) / (`Xmit Bytes/s + Recv Bytes/s`)).

Average packet size is a tuning indicator. Tuning the network by increasing the MTU size is not effective if applications use small packets.

The Detailed Network Device-Driver Statistics report that netpmon produces is shown in Figure 86. The report displays CPU service time statistics of network packets.

```
Detailed Network Device-Driver Statistics:
------------------------------------------

DEVICE: ethernet 0
recv packets:           9351
  recv sizes (bytes):   avg 68.4    min 60      max 442     sdev 24.9
  recv times (msec):    avg 0.018   min 0.011   max 0.103   sdev 0.004
  demux times (msec):   avg 16.335  min 0.095   max 149429.476 sdev 1548.759
xmit packets:           16987
  xmit sizes (bytes):   avg 1484.0  min 60      max 1514    sdev 204.3
  xmit times (msec):    avg 1192.549 min 0.056   max 15717.099 sdev 509.452
```

Figure 86. Network Adapter Utilization Detailed Report

11.5.5.4 Identification of Outgoing Network Traffic

The Network Device-Driver Transmit Statistics (by Destination Host) report produced by netpmon displays network transmission to each destination.

Use this report to assess the traffic generated by this host and where information was sent. Use this information to look for candidate systems on the network that may be bottlenecks.

In Figure 87 on page 211 the traffic flow is mainly to node 5 (sp5n05.msc.itso.ibm.com).

```
Network Device-Driver Transmit Statistics (by Destination Host):
----------------------------------------------------------------

Host                     Pkts/s  Bytes/s
----------------------------------------
sp5n05.msc.itso.ibm.com   90.22   135219
sp5en0.msc.itso.ibm.com    0.94       65
```

Figure 87. Network Transmission Summary Report

Each host listed in the Network Device-Driver Transmit Statistics (by Destination Host) report also appears in the Detailed Network Device-Driver Transmit Statistics (by Host) report which netpmon produces.

Using this report, CPU service times for each packet sent can be determined.

An example of the Detailed Network Device-Driver Transmit Statistics (by Host) report is shown in Figure 88.

```
Detailed Network Device-Driver Transmit Statistics (by Host):
-------------------------------------------------------------

HOST: sp5n05.msc.itso.ibm.com
xmit packets:          16811
  xmit sizes (bytes):   avg 1498.8  min 66      max 1514    sdev 144.8
  xmit times (msec):    avg 1204.961 min 0.088   max 15717.099 sdev 497.306

HOST: sp5en0.msc.itso.ibm.com
xmit packets:          175
  xmit sizes (bytes):   avg 69.4    min 60      max 699     sdev 51.9
  xmit times (msec):    avg 7.084   min 0.056   max 1143.339 sdev 86.336
```

Figure 88. Network Transmission Detailed Report

11.5.5.5 Identification of Network Service Calls

The Socket Call Statistics (by Process) report produced by netpmon is a breakdown of socket service requests by protocol issues in each process.

Use this report to asses the type of network traffic each application uses.

Figure 89 on page 212 shows an example of the Socket Call Statistics (by Process) report.

```
TCP Socket Call Statistics (by Process):
----------------------------------------
                                   ------ Read -----   ----- Write -----
Process (top 20)             PID   Calls/s   Bytes/s   Calls/s   Bytes/s
------------------------------------------------------------------------
fault_service_Worm_RTG_SP  16662      0.30         1      0.06         2
SDRGetObjects              19022      0.04         1      0.01         0
SDRGetObjects              19016      0.04         1      0.01         0
------------------------------------------------------------------------
Total (all processes)                 0.37         3      0.07         2

========================================================================

ICMP Socket Call Statistics (by Process):
-----------------------------------------
                                   ------ Read -----   ----- Write -----
Process (top 20)             PID   Calls/s   Bytes/s   Calls/s   Bytes/s
------------------------------------------------------------------------
ping                       19026      0.03         6      0.03         2
ping                       19024      0.03         5      0.03         2
ping                       19020      0.00         0      0.02         1
------------------------------------------------------------------------
Total (all processes)                 0.06        11      0.07         5
```

Figure 89. Network Process Service Calls Summary Report

A Detailed TCP Socket Call Statistics (by Process) report is produced by netpmon for each process listed in the Socket Call Statistics (by Process) report.

Use this report to estimate each application’s network CPU requirement.

This report is of most value when calculating an application’s network resources and the resulting CPU resource requirement.

Figure 90 on page 213 shows an example of the Detailed TCP Socket Call Statistics (by Process) report.

```
Detailed TCP Socket Call Statistics (by Process):
-------------------------------------------------

PROCESS: fault_service_Worm_RTG_SP   PID: 16662
reads:                 56
  read sizes (bytes):   avg 2.5     min 1       max 26      sdev 5.6
  read times (msec):    avg 0.611   min 0.021   max 3.287   sdev 1.110
writes:                12
  write sizes (bytes):  avg 27.7    min 5       max 64      sdev 21.0
  write times (msec):   avg 0.200   min 0.180   max 0.266   sdev 0.025

PROCESS: /usr/lpp/ssp/bin/SDRGetObjects   PID: 19022
reads:                 7
  read sizes (bytes):   avg 28.9    min 1       max 196     sdev 68.2
  read times (msec):    avg 0.473   min 0.018   max 3.004   sdev 1.034
writes:                1
  write sizes (bytes):  avg 23.0    min 23      max 23      sdev 0.0
  write times (msec):   avg 0.263   min 0.263   max 0.263   sdev 0.000

PROCESS: /usr/lpp/ssp/bin/SDRGetObjects   PID: 19016
reads:                 7
  read sizes (bytes):   avg 28.9    min 1       max 196     sdev 68.2
  read times (msec):    avg 0.447   min 0.018   max 2.855   sdev 0.984
writes:                1
  write sizes (bytes):  avg 23.0    min 23      max 23      sdev 0.0
  write times (msec):   avg 0.263   min 0.263   max 0.263   sdev 0.000

PROTOCOL: TCP (All Processes)
reads:                 70
  read sizes (bytes):   avg 7.8     min 1       max 196     sdev 32.7
  read times (msec):    avg 0.581   min 0.018   max 3.287   sdev 1.093
writes:                14
  write sizes (bytes):  avg 27.0    min 5       max 64      sdev 19.5
  write times (msec):   avg 0.209   min 0.180   max 0.266   sdev 0.032
```

Figure 90. Network Process Service Calls Detailed Report

11.5.5.6 Identification of Network File System Activity

NFS servers usually have very high network traffic. They are often productive hunting grounds in the search for performance problems especially when searching for network performance problems.

The NFS Server Statistics (by Client) report produced by netpmon provides NFS activity statistics for each NFS client.

Use this report to assess the network load of NFS.

Figure 91 on page 214 shows an example of the NFS Server Statistics (by Client) report, showing NFS activity for the client node sp5n13.

```
NFS Server Statistics (by Client):
----------------------------------
                         ------ Read -----   ----- Write -----     Other
Client                   Calls/s   Bytes/s   Calls/s   Bytes/s   Calls/s
------------------------------------------------------------------------
sp5sw13.msc.itso.ibm.com    0.00         0      0.00         0      3.87
------------------------------------------------------------------------
Total (all clients)         0.00         0      0.00         0      3.87
```

Figure 91. Utilization of Network by NFS Summary Report

Each NFS client listed in the netpmon NFS Server Statistics (by Client) report is listed in the Detailed NFS Server Statistics (by Client) report, which provides a breakdown of NFS service times.

Figure 92 is an example of the Detailed NFS Server Statistics (by Client) report produced by netpmon.

```
Detailed NFS Server Statistics (by Client):
-------------------------------------------

CLIENT: sp5sw13.msc.itso.ibm.com
other calls:            732
  other times (msec):   avg 4.226   min 0.356   max 481.851 sdev 18.283

COMBINED (All Clients)
other calls:            732
  other times (msec):   avg 4.226   min 0.356   max 481.851 sdev 18.283
```

Figure 92. Utilization of Network by NFS Detailed Report

11.5.6 Checking Network Adapter Settings Using lsattr

This command displays adapter settings.

Use it to check the following network adapter settings:

- Transmit queue lengths
- Maximum Transmission Unit (MTU)
- Memory pool sizes

Figure 93 on page 215 shows an example of using lsattr to obtain the adapter settings for css0.

```
# lsattr -El css0
bus_mem_addr   0x04000000 Bus memory address        False
int_level      0xb        Bus interrupt level       False
int_priority   3          Interrupt priority        False
dma_lvl        8          DMA arbitration level     False
spoolsize      524288     Size of IP send buffer    True
rpoolsize      524288     Size of IP receive buffer True
adapter_status css_ready  Configuration status      False
```

Figure 93. Listing Adapter Attributes with lsattr

Note

Each adapter has a different set of attributes. Figure 93 is only an example of using `lsattr` to check adapter attributes.

11.5.7 Tuning Network Parameters Using no

This command sets the kernel network parameters.

This is the network tuning command that is used most often. Changes remain active only until the system is rebooted.

Exercise care when setting network options. This command does not range-check the values entered, and it is possible to make your system inoperative. Always perform a sanity check (do not, for example, set a maximum value that is less than the corresponding minimum value). If you do not understand the purpose of a network attribute, find out what it does before changing it.

To display a network attribute, use no -o attribute. For example, to display the value of thewall, use no -o thewall.

To set the value of a network attribute, use no -o attribute=value. For example, to set the value of thewall to 65536, use no -o thewall=65536.

Before setting network attributes, check the current values. Use no -a to list all network attributes.

To set a network attribute back to the default value, use no -d attribute. For example, to set the network attribute thewall back to the default value, use no

-d thewall. Default values are often not optimal for the RS/6000 SP, but they do provide a stable network environment.

Place changes made to network tunables via `no` into /tftpboot/tuning.cust except changes to the Address Resolution Protocol (ARP) parameters. These must be placed into /etc/rc.net. Placing changes in these files insures that the next time the system is booted, the changes will remain in effect.

We will not cover all the network attributes that are set using `no`. We only discuss the network attributes that we have found play an important role in network optimization on the RS/6000 SP.

11.5.7.1 Setting Upper Memory Bound for Communications

The network attribute thewall sets the upper bound on memory that can be used by the communications subsystem. The value is specified in KB.

Example:

```
no -o thewall=65536
```

This sets the maximum amount of memory allocated to the communications subsystem to 64 MB.

Changes to this attribute take effect immediately for new connections.

11.5.7.2 Setting Upper Bounds for Network Buffers

To limit the memory that all network buffers use, set sb_max. This value is specified in bytes.

Do not set this attribute to a value larger than the network attribute thewall.

Set the limits of the send and receive buffers for each socket connection in bytes using the following network attributes:

udp_sendspace	Sets the maximum size of each UNIX Domain Protocol (UDP) socket send buffer. This value must be less than or equal to sb_max.
tcp_sendspace	Sets the maximum size of each TCP socket send buffer. This value must be less than or equal to sb_max.
upd_recvspace	Sets the maximum size of each UDP socket receive buffer. This value must be less than or equal to sb_max.
tcp_recvspace	Sets the maximum size of each TCP socket receive buffer. This value must be less than or equal to sb_max.

We recommend, for the RS/6000 SP, that the largest socket buffer should not exceed half the value assigned to the network attribute sb_max.

We also recommend that the buffers should be larger than the largest MTU.

The network attribute ipqmaxlen sets the maximum number of packets that can be held in the input queue.

Example:

```
no -o sb_max=2097152 -o tcp_sendspace=524288 -o tcp_recvspace=524288 -o
ipqmaxlen=512
```

Set the maximum upper memory bound to 2 MB, set the send/receive buffers limit for each TCP socket to 512 KB, and set the input queue length to 512 packets.

With the exception of ipqmaxlen, changes take effect immediately for new connections. Changes to ipqmaxlen take effect when the system is rebooted.

11.5.7.3 Setting Network Address Resolution Attributes

The ARP network attributes that need to be set are:

arptab_bsiz	Sets the number of entries in each ARP table. We recommend that this value be twice the number of network adapters, and never less than the default value of 7.
arptab_nb	Sets the number of ARP tables.
arpqsiz	Sets the number of network packets that can be queued for an ARP response.

Note

Changes to arptab_bsiz and arptab_nb will not take effect until the system is rebooted.

Two other network address resolving attributes that often need changing are:

subnetsarelocal	When set, specifies that any network packets with addresses matching the local area network mask are local addresses.
ipforwarding	When set, enables network address forwarding to gateway hosts for resolution.

Note

These changes take effect immediately for new connections.

Example:

```
no -o arptab_bsiz=7 -o arptab_nb=25 -o ipforwarding=1
```

Set the number of entries in each of the 25 ARP tables to **7**, so that network addresses can be forwarded to a gateway host for resolution.

11.5.7.4 Setting Transmission Unit Sizes

The network attribute tcp_mssdflt is the default maximum MTU size used when communicating through other hosts to clients. Often, it is set to a pessimistically small value, which does not make efficient use of network adapters with large MTUs. If set too large, inefficiencies result as the packets are fragmented before being forwarded onto the client.

The network attribute tcp_pmtu_discover, when set, attempts to discover the largest MTU that can be routed through the network to a destination. This overcomes the problem of determining an appropriate value for tcp_mssdflt. It creates a performance problem if the routing table becomes large.

Note

Not all systems support tcp_pmtu_discover. If it is used, and the host or device targeted does not support this feature, tcp_mssdflt is used to set the MTU size.

Example:

```
no -o tcp_mssdflt=1448 -o tcp_pmtu_discover=0
```

This sets the default MTU size to 1448 bytes, and turns off MTU discovery.

11.5.7.5 Enabling TCP Enhancements for High Performance

TCP enhancements for high performance are enabled by setting the network attribute rfc1323. Use this network option for all nodes in an RS/6000 SP complex if the SP Switch is installed.

Setting rfc1323 enables the following TCP extensions:

- An increase of the upper limit for tcp_sendspace to 4 GB, up from 64 KB
- An increase of the upper limit for tcp_recvspace to 4 GB, up from 64 KB
- An increase of the maximum window size to 4 GB, up from 64 KB

- Time-stamped acknowledgments for better round-trip estimations
- Protection against wrapped sequence numbers

If rfc1323 is not set, the RS/6000 SP complex does not take advantage of the high switch bandwidth when using TCP.

Example:

```
no -o rfc1323=1
```

This enables the TCP extensions.

11.5.8 Tuning NFS Network Parameters Using nfso

Use this command to configure the NFS attributes.

Except for nfs_socketsize, which requires NFS to be stopped and restarted, NFS network attributes set with this command take effect immediately.

Changes made using nfs0 are lost when the system is rebooted unless the changes are placed in /tftpboot/tuning.cust.

To view an NFS attribute, use no -o attribute. For example, to view the attribute nfs_socketsize, use nfso -o nfs_socketsize.

To set an NFS attribute, use no -o attribute=value. For example, nfso -o nfs_socketsize=262144 sets the NFS attribute nfs_socketsize to 262144.

Do not change an NFS attribute unless you understand the meaning and effect of changing it.

Always exercise care when changing NFS network attributes with nfso. No range checking is done. Setting these attributes incorrectly can cause a system crash or make NFS inoperative.

To obtain a list of all NFS attributes, use no -a.

NFS attributes can be reset to default values by using the no -d attribute. For example, nfso -d nfs_socketsize sets nfs_socketsize to the default value. The default values are not optimum, but they do provide a stable system.

We will not cover all of the network attributes that can be set using nfso, we will only discuss the NFS attributes that we have found play an important role in optimizing NFS on the RS/6000 SP.

For a discussion and the suggested values for NFS attributes, refer to Chapter 8, “Global File Systems Tuning” on page 89.

11.5.8.1 Setting NFS Network Buffers

Set the send and receive buffers using the following NFS attributes:

nfs_socketsize: Sets the maximum size of the UDP send and receive buffers in bytes.

nfs_tcp_socketsize: Sets the maximum size of the TCP send and receive buffers in bytes.

nfs_device_specific_bufs: When set, enables NFS to use device-allocated memory buffers if supported by the network adapter. We recommend that this option should always be enabled.

The size of the NFS socket buffers must be less than the network attribute sb_max. Refer to 11.5.7.2, “Setting Upper Bounds for Network Buffers” on page 216 for a description of sb_max.

Example:

```
nfso -o nfs_socketsize=262144 -o nfs_tcp_socketsize=262144 -o
nfs_device_specific_bufs=1
```

This sets NFS socket buffer sizes to 256 KB and enables device memory allocation of NFS buffers if supported by the adapter.

11.5.8.2 Enabling NFS TCP Enhancements for High Performance

When NFS is used with the high-speed switch, the TCP enhancements for high performance must be enabled. Set nfs_rfc1323 to enable TCP high performance enhancements.

For further information about the TCP high performance enhancements, refer to 11.5.7.5, “Enabling TCP Enhancements for High Performance” on page 218.

Example:

```
nfso -o nfs_rfc1323=1
```

Enables TCP high performance network enhancements for NFS.

11.5.9 Tuning Network Switch Parameters Using chgcss

This command is used to change the kernel switch attributes.

Use chgcss to change the size of the switch receive and send buffer pool size. We recommend that the receive and send buffer pool size are larger than the network attribute "sb_max" (refer to 11.5.7.2, “Setting Upper Bounds for Network Buffers” on page 216).

The switch buffer pools are defined using the following switch attributes:

rpoolsize: Sets the receive buffer pool size.

spoolsize: Sets the send buffer pool size.

The switch send and receive buffer pools are pinned in memory. When sending data, if the pool size is exceeded, temporary memory is allocated to handle the overflow. This affects performance.

Example:

```
chgcss -l css0 -a rpoolsize=2097152 -a spoolsize
```

Set the receive and send buffer pools to 2 MB for the switch adapter css0.

Changes made to switch settings do not take effect until the switch is restarted. Restart the system or do the following:

1. Efence node
2. ifconfig css0 down
3. ifconfig css0 detach
4. chgcss -l css0 -a rpoolsize=value -a spoolsize=value
5. rc.switch
6. Eunfence node (for HiPS switches use Estart)

Changes made to the switch parameters using chgcss are stored permanently, but it is still a good idea to enter these changes into /tftpboot/tuning.cust.

11.6 Investigation

We use the following steps when looking for performance problems:

1. Check for an I/O problem using iostat. Refer to 11.4.1, “Monitoring I/O Using iostat” on page 179.
2. Check for a memory problem using vmstat. Refer to 11.2.1, “Monitoring Memory with vmstat” on page 156.
3. Check for a CPU problem using sar. Refer to 11.3.1, “Monitoring the CPU with vmstat” on page 167.

4. Check for a network problem using netstat. Refer to 11.5.3, “Monitoring the Network with netstat” on page 199.
5. Check for an adapter problem using the adapter stat commands entstat, tokstat, fddistat, atmstat, and estat. Refer to 11.5.1, “Monitoring the Network Using Adapter Statistics” on page 195.
6. Check for a switch problem using vdidl2 or vdidl3. Refer to 11.5.2, “Monitoring the Switch with vdidl2 or vdidl3” on page 197.
7. Check which processes are running and their resource requirements using ps. Refer to 11.2.4, “Monitoring Memory with ps” on page 159, and 11.3.3, “Monitoring the CPU Using ps” on page 170.
8. Check for a paging problem using lsps. Refer to 11.2.3, “Monitoring Memory with lsps” on page 158.

11.7 Performance Toolbox for AIX (PTX/6000)

The performance toolbox for AIX (PTX/6000) is a tool to monitor and tune system performance.

PTX/6000 uses a client/server model to monitor local and remote systems. It presents the performance data graphically.

Figure 94 on page 223 illustrates the client/server model used by PTX/6000. The server requests performance data from the nodes or other networked computers. The nodes or networked computers in turn supply a stream of performance data. The server displays the performance data graphically. When monitoring is complete, the server informs the nodes or networked computers to stop streaming performance data.

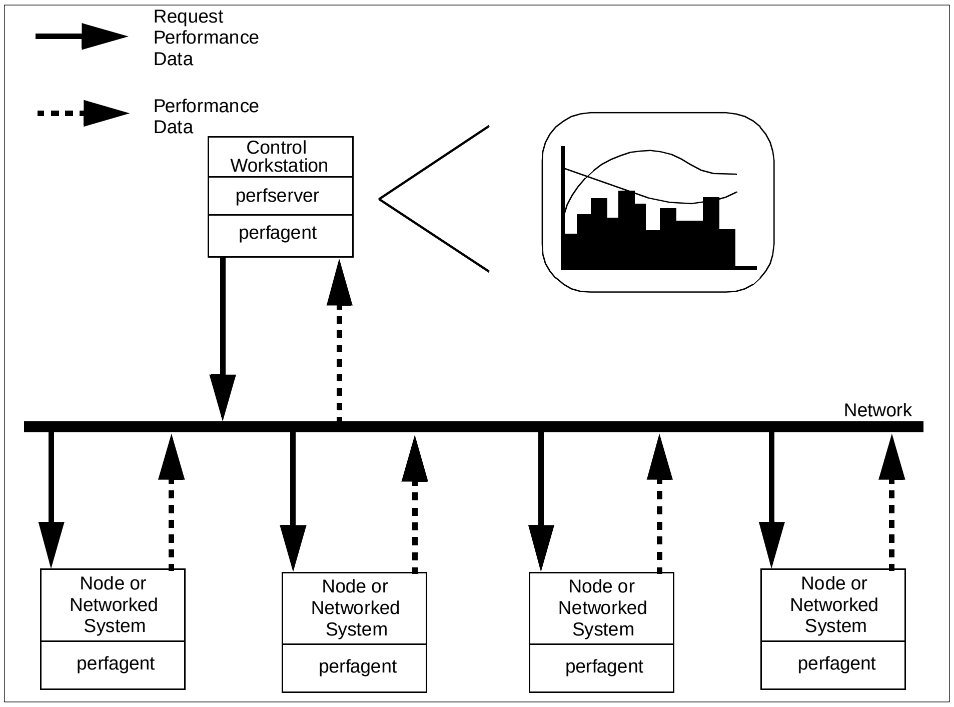


Figure 94. PTX/6000 Network Monitoring - Client/Server Model

The ability to monitor local or remote systems and the network is important in an RS/6000 SP environment where a large number of nodes need to be monitored using a single point of control.

PTX/6000 has the following features:

- It monitors system resources.
- It analyzes system resource statistics.
- It uses a Graphical User Interface (GUI).
- It is able to process Simple Network Management Protocol (SNMP) requests.
- It supports the RS/6000 SP Performance Toolbox Parallel Extensions (PTPE).
- It has an Application Programming Interface (API) to create custom-written applications for analysis of performance archives.

11.7.1 PTX/6000 Installation

In an RS/6000 SP environment, we recommend that PTX/6000 be installed on the control workstation and any nodes which are to be monitored or managed.

We also recommend that the control workstation be used as a PTX/6000 server. The control workstation is designed to be a single point of management. Using a node introduces another point of management and adds an additional system overhead.

11.7.1.1 Installation of PTX/6000 on the Control Workstation

The steps we used to install PTX/6000 on our control workstation were:

1. Create a PTX/6000 install directory:

   ```
   mkdir -p /spdata/sys1/install/sp4/ptx
   ```

2. Change the working directory to the install directory:

   ```
   cd /spdata/sys1/install/sp4/ptx
   ```

3. Load the PTX/6000 file sets into the install directory. The method used will vary depending on the location and media chosen. We used ftp to retrieve the file sets from an ftp server.

   The file sets required are:

   - perfagent.server: Performance agent and daemons
   - perfagent.tools: Local performance analysis and control commands
   - perfmgr: PTX/6000 performance monitoring software
   - perfmgr.local: PTX/6000 local monitoring software
   - perfmgr.network: PTX/6000 remote monitoring software

4. Install the PTX/6000 performance management software:

   ```
   installp -aXd /spdata/sys1/install/sp4/ptx perfmgr
   ```

5. Install the PTX/6000 local monitoring software:

   ```
   installp -aXd /spdata/sys1/install/sp4/ptx perfmgr.local
   ```

6. Install the PTX/6000 network monitoring software:

   ```
   installp -aXd /spdata/sys1/install/sp4/ptx perfmgr.network
   ```

7. Verify the installation:

   ```
   lslpp -l perfmgr.*
   ```

8. From the delivery of PSSP 2.2 onwards, the PTX/6000 agent software is a prerequisite. It should, therefore, already be installed.

To check if it is installed, use lslpp -l peragent.*.

If it has not been installed, use the following steps to install the PTX/6000 agent software:

1. Install the PTX/6000 remote agent:

   ```
   installp -aXd /spdata/sys1/install/sp4/ptx perfagent.server
   ```

2. Install the PTX/6000 remote agent tools:

   ```
   installp -aXd //spdata/sys1/install/sp4/ptx perfagent.tools
   ```

3. Verify the installation:

   ```
   lslpp -l perfagent.*
   ```

9. Export the install directory from the control workstation to the nodes:

   ```
   /usr/sbin/mknfsexp -d '/spdata/sys1/install' -t 'rw' \
   -r '<sp node list>' 'B'
   ```

Note

You must be logged on as root to install PTX/6000.

11.7.1.2 Installation of PTX/6000 on Each Node

To check if the perfagent software has already been installed, use lslpp -l perfagent.*. If perfagent.server or perfagent.tools are not installed, the following steps will install them:

1. Create an NFS mount directory for the software installation:

   ```
   mkdir -p /install
   ```

2. NFS-mount the installation directory:

   ```
   mount <control workstation>:/spdata/sys1/install/sp4 /install
   ```

3. Install the PTX/6000 remote agent:

   ```
   installp -aXd /install/ptx perfagent.server
   ```

4. Install the PTX/6000 remote agent tools:

   ```
   installp -aXd /install/ptx perfagent.tools
   ```

5. Verify the installation:

   ```
   lslpp -l perfagent.*
   ```

Note

You must be logged on as root to install PTX/6000.

11.7.2 Using PTX/6000 to Monitor an RS/6000 Cluster

The PTX/6000 performance manager is started from the command line with xmperf.

Performance monitoring has been set up by using a hierarchal relationship between the performance monitoring objects. Figure 95 shows the monitoring relationships.

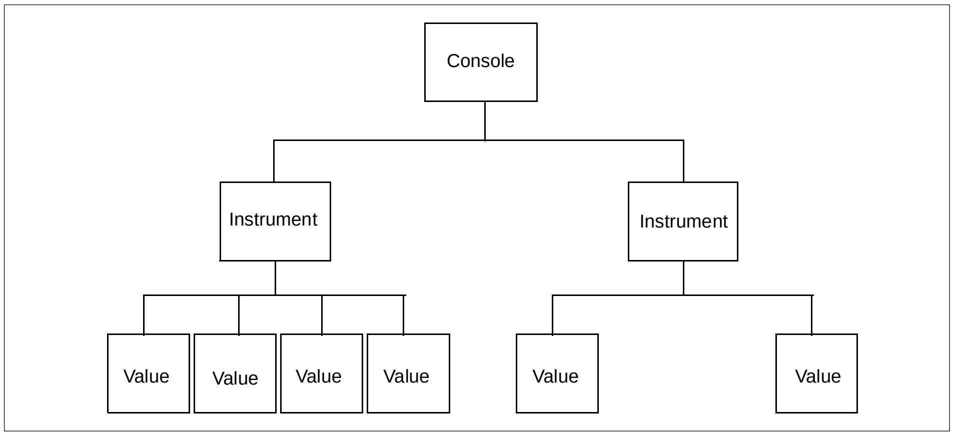


Figure 95. PTX/6000 Monitoring Presentation Component Hierarchy

A console is a frame that encompasses performance monitoring instruments. PTX/6000 provides default consoles. Use the default consoles as templates when creating new consoles.

An instrument is a frame that displays performance data. An instrument defines the type of display used to present the performance data. The instruments supplied are:

- Area graph
- Bar graph
- Line graph
- Pie chart
- Skyline graph
- Speedometer
- State bar
- State light

A value is a performance statistic that is displayed by an instrument. A performance instrument can display multiple values. Depending on the type of instrument, the values are displayed and stacked differently.

Figure 96 on page 227 shows an example of using the PTX/6000 GUI interface xmperf to monitor an RS/6000 SP control workstation.

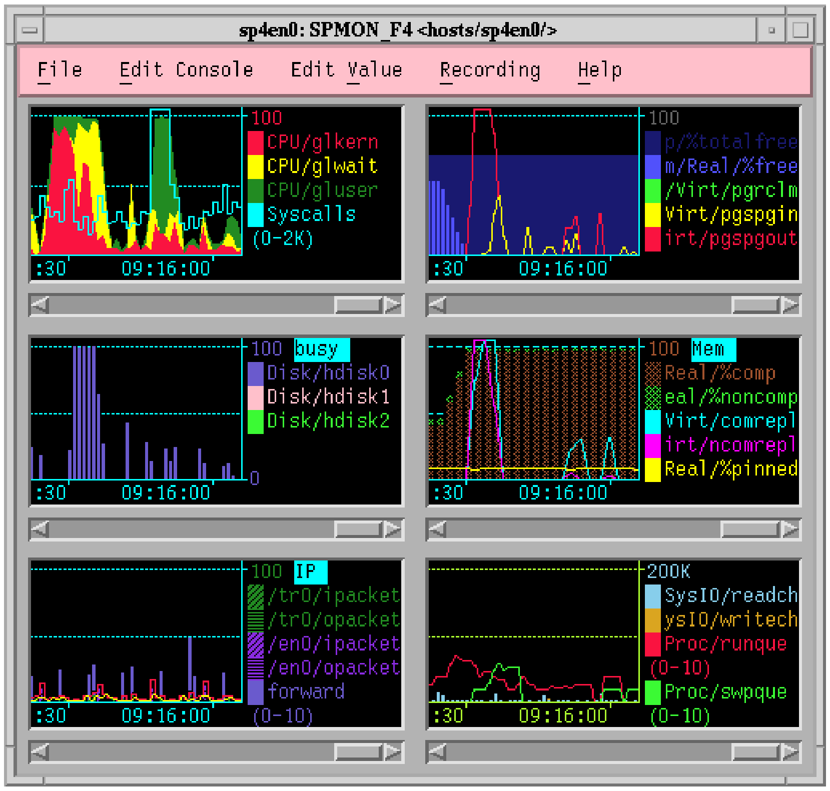


Figure 96. Monitoring a System Using PTX/6000

We created a console (SPMON_F4 shown in Figure 96) that encapsulated the six elements we recommend should be monitored:

1. CPU
   - Kernel CPU time
   - CPU I/O wait
   - User CPU time
   - System call CPU time
2. VMM
   - Percentage of page space free
   - Percentage of real memory free
   - Page reclaims

- Page in count
- Page out count

3. Disk activity
   - Utilization of hdisk0
   - Utilization of hdisk1
   - Utilization of hdisk2

4. Memory
   - Percentage of real memory computational pages
   - Percentage of real memory noncomputational pages (file pages)
   - Percentage of total memory computational
   - Percentage of total memory noncomputational (file pages)
   - Percentage of real memory pinned (nonswappable)

5. Network
   - Packets received on adapter tr0
   - Packets sent on adapter tr0
   - Packets received on adapter en0
   - Packets sent on adapter en0
   - Packets forwarded
   - Total packets received
   - Datagrams sent

6. I/O kernel calls
   - Number of bytes read
   - Number of bytes written
   - Size of run queue
   - Number of runnable processes waiting to be paged in

We observed that, with a tool like PTX/6000, the initial reaction is often to display as much data as possible. This leads to an aesthetically unpleasing appearance, where critical performance figures are hidden by details.

We recommend that a high-level system monitoring console similar to the example shown in Figure 96 on page 227 be created. Use this console to display performance values that, in your environment, are key elements in determining when system performance and throughput are unacceptable.

For further information about PTX/6000, refer to *Performance Toolbox Version 1.2 and 2 for AIX: Guide and Reference,* SC23-2625.

11.8 Performance Toolbox Parallel Extensions (PTPE)

PTPE is an extension of the Performance Toolbox (PTX/6000) for use in an RS/6000 SP complex.

PTPE is a scalable performance monitor, and when installed in an RS/6000 SP complex, it provides easy access to performance information.

Any node or group of nodes can be monitored with a single point of control. The performance parameters and how the information is to be displayed are definable. Monitoring can be in real time, or the performance data can be archived for later analysis.

PTPE performance statistics can be viewed as follows:

- On a per-node basis
- Averaged across a group of nodes
- Averaged across an RS/6000 SP complex

PTPE setup, configuration and management has been integrated into Perspectives. PTPE also supports configuration and setup using AIX commands.

An API is provided. This allows development of customized applications to extract information from a PTPE archive.

11.8.1 PTPE Installation

PTPE must be installed on the control workstation and on each node.

Note

Install PTPE on the control workstation *before* installing PTPE on the nodes.

The prerequisites before installing PTPE are:

1. AIX version 4 or later.
2. RS/6000 Cluster Technology.
3. Performance Aide for AIX must be installed on the control workstation and all nodes that will be monitored.
4. Performance Toolbox (PTX/6000) must be installed on any node or control workstation that will be used to display or analyze the performance data.

11.8.1.1 Installation of PTPE on the Control Workstation

The steps we used to install PTPE on the control workstation were:

1. Create a PSSP install directory:

   ```
   mkdir -p /spdata/sys1/install/pssplpp/PSSP-3.1
   ```

> **Note**
>
> The name of the PSSP directory will depend on the version of PSSP that is used. We used PSSP version 3.1. The PSSP install directory was therefore called PSSP-3.1.

2. Change the working directory to the PSSP install directory:

   ```
   cd /spdata/sys1/install/pssplpp/PSSP-3.1
   ```

3. Load the PSSP software into the PSSP install directory. The method used will vary depending on the location and media chosen. We used ftp to retrieve the PSSP software from an ftp server.
4. Install the PTPE programs:

   ```
   installp -aXd /spdata/sys1/install/pssplpp/PSSP-3.1 ptpe.program
   ```

5. Install the PTPE documentation:

   ```
   installp -aXd /spdata/sys1/install/pssplpp/PSSP-3.1 ptpe.docs
   ```

6. Verify the installation:

   ```
   lslpp -l pte*
   ```

7. Register the PTPE spdmd daemon:

   ```
   /usr/lpp/ptpe/bin/spdmdctrl -a
   ```

8. Export the PSSP install directory from the control workstation to the nodes:

   ```
   /usr/sbin/mknfsexp -d '/spdata/sys1/install/pssplpp' -t 'rw' \
   -r '<sp node list>' 'B'
   ```

9. Create a PTPE monitoring group:

   ```
   /usr/lpp/ptpe/bin/ptpegroup
   ```

> **Note**
>
> You must be logged in as root to install PTPE.

11.8.1.2 Installation of PTPE on Each Node

The steps we used to install PTPE on each node were:

1. Create an NFS mount directory for the PSSP software installation:

   ```
   mkdir -p /install_pssp
   ```

2. NFS-mount the PSSP install directory:

   ```
   mount <control workstation>:/spdata/sys1/install/pssplpp /install_pssp
   ```

3. Install the PTPE programs:

   ```
   installp -aXd /install_pssp/PSSP-3.1 ptpe.program
   ```

4. Verify the installation:

   ```
   lslpp -l pte*
   ```

5. Add a supplier resource entry:

   ```
   cp /usr/samples/perfagent/server/xmservd.res /etc/perf
   echo "supplier: /usr/lpp/ptpe/bin/ptpertm -p" >> /etc/perf/xmservd.res
   ```

6. Register the PTPE spdmd daemon:

   ```
   /usr/lpp/ptpe/bin/spdmdctrl -a
   ```

Note

You must be logged in as root to install PTPE.

11.8.1.3 PTPE Monitoring Hierarchy

The collection of performance monitoring data is centralized through a monitoring hierarchy.

Distributing the management of performance information across a number of nodes using a performance monitoring hierarchy circumvents the inherent limitation of simultaneous monitoring of a large number of nodes.

PTPE Monitoring Hierarchy Node Classification

The following PTPE classifications are used to describe the roles of nodes in the hierarchy:

1. Central coordinator

   One node is assigned the role of central coordinator. This node coordinates and administers data managers.

   The responsibilities of the central coordinator are:

   - Calculate average statistics from the performance summaries provided by the data managers. This provides a system-wide performance summary.

- Manage the performance data collection and archiving activities of the data managers.

  The central coordinator informs the data manager when to instruct the nodes to start or stop collecting performance data. It also informs the data managers when to instruct the nodes to archive performance data.

- Route all PTPE API requests.

2. Data managers

   In a PTPE hierarchy, nodes are assigned data management responsibility for other nodes. These data management nodes coordinate the collection of performance data from the nodes reporting to them.

Note

The central coordinator must be managed by a data manager.

   The responsibilities of the data manager are:

   - Calculate average statistics from the performance summaries provided by each node in the group. This provides a group performance summary.
   - Manage the collection and archiving of performance data within the group as directed by the central coordinator.
   - Routes requested performance data from the nodes managed back to the requester.

3. Reporters

   Reporters collect performance data and report this data to their data managers.

Note

Data managers are reporters and report to themselves or another data manager. The central coordinator is also a reporter.

   The responsibilities of reporters are:

   - Calculate average statistics from the performance data.
   - Collect performance data.
   - Provide requested performance data back to their data managers.

4. Non-Participants

   Nodes that do not participate in the PTPE performance hierarchy do not report performance data.

Planning a PTPE Monitoring Hierarchy

Several considerations should guide the planning of a PTPE monitoring hierarchy:

1. Which nodes can accept increased network traffic?

Managers and the central coordinator should be chosen from nodes that fall into this group.

2. Which nodes can be grouped together to yield the most meaningful performance summaries?

   Nodes that can be grouped and will give meaningful indications of system activity should be grouped together under a single data manager.

   Examples of logical groups:

   - Database servers
   - File servers
   - Application servers
   - Gateways

3. The number of nodes reporting to each data manager.

   The more nodes that report to each data manager, the greater the performance impact.

   Fewer nodes reporting to each data manager reduces the additional overhead for the data managers, but it increases the effort required to manage the complex.

   For example, if all database servers report to one data manager, a single point of reference to identify the database server performance is achieved.

4. Should all nodes be monitored?

   Nodes with dedicated tasks, or nodes that require all of their resources may need to run without the additional overhead of collecting and reporting performance data.

Designing and Documenting a PTPE Monitoring Hierarchy
Before implementing a PTPE monitoring hierarchy, we recommend that you create a written record of the PTPE performance monitoring hierarchy. This record should then be included with the system's site documentation.

An up-to-date written record of the PTPE monitoring hierarchy is a good reference when looking at a system's performance characteristics, or when adding new nodes.

As an example, we set up PTPE 3.1 and documented the PTPE performance hierarchy on an RS/6000 SP frame with 10 nodes. The frame was configured as shown in Figure 97 on page 235.

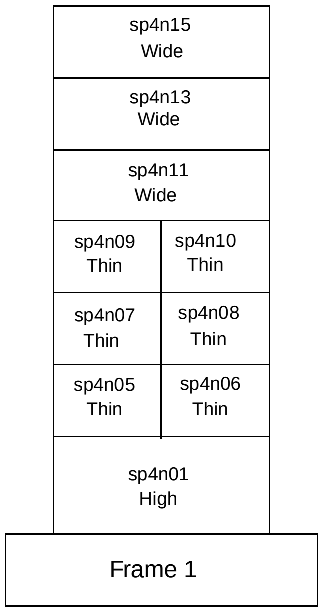


Figure 97. Example of an RS/6000 SP Frame Configuration

Node `sp4n01` is an eight-way High node, which we decided would be the central coordinator.

Nodes `sp4n05` and `sp4n10` we designated as data managers. Each data manager would manage five nodes. The PTPE performance hierarchy we used is shown in Figure 98 on page 236.

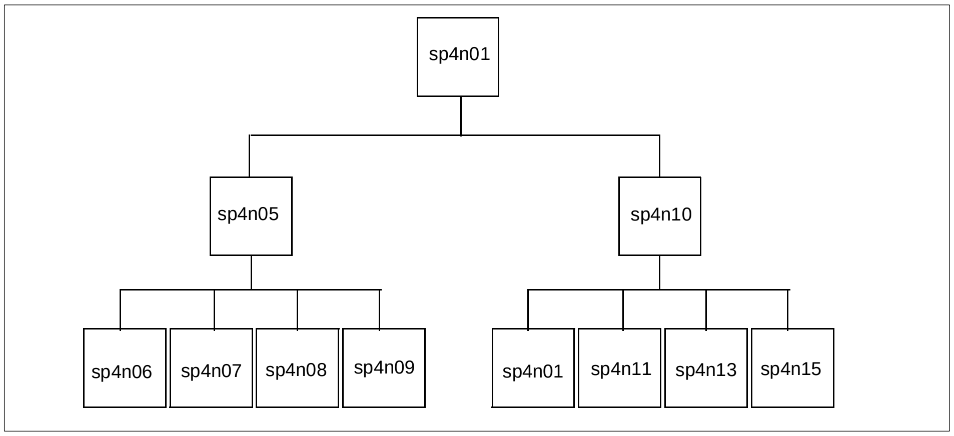


Figure 98. PTPE Node Hierarchy

With the planning of our PTPE monitoring hierarchy complete, the final step was to document the PTPE performance hierarchy.

We produced a table showing the responsibilities each node would have in the PTPE performance hierarchy and which node they would report to. Table 27 is the documentation produced for the PTPE performance hierarchy shown in Figure 98.

Table 27. Documenting a PTPE Performance Hierarchy

Frame	Node	Name	Role	Manager
1	1	sp4n01	Central Coordinator	sp4n10
1	5	sp4n05	Data Manager	
1	6	sp4n06	Reporter	sp4n05
1	7	sp4n07	Reporter	sp4n05
1	8	sp4n08	Reporter	sp4n05
1	9	sp4n09	Reporter	sp4n05
1	10	sp4n10	Data Manager	
1	11	sp4n11	Reporter	sp4n10
1	13	sp4n13	Reporter	sp4n10
1	15	sp4n15	Reporter	sp4n10

Note

All nodes are participants in our PTPE performance hierarchy.

Installing a PTPE Monitoring Hierarchy

To create a PTPE performance hierarchy on the control workstation, use the Perspectives performance manager, or the command `ptpehier`.

We created a UNIX file, /etc/ptpe_hierarchy, to define the PTPE performance hierarchy definition.

Once we had defined the PTPE performance hierarchy, we used `ptpehier` to add the PTPE performance monitoring to the system.

With the PTPE performance monitoring hierarchy in place, we started the PTPE data collection process using `ptpectrl`.

Example:

The `ptpehier` command requires the PTPE performance hierarchy to be defined as a series of node groups each with a designated node leader:

- Each node group starts with {.
- The first node listed is the group leader (data manager).
- Other nodes listed are group members.
- Each node group ends with }.

Using the PTPE performance hierarchy plan shown in Table 27 on page 236, we created the file /etc/ptpe_hierarchy using vi as shown in Figure 99 to define the PTPE performance monitoring hierarchy.

```
{
sp4n05.msc.itso.ibm.com
sp4n06.msc.itso.ibm.com
sp4n07.msc.itso.ibm.com
sp4n08.msc.itso.ibm.com
sp4n09.msc.itso.ibm.com
}
{
sp4n10.msc.itso.ibm.com
sp4n01.msc.itso.ibm.com
sp4n11.msc.itso.ibm.com
sp4n13.msc.itso.ibm.com
sp4n15.msc.itso.ibm.com
}
```

Figure 99. PTPE Performance Hierarchy Definition

In Figure 99 on page 238, sp4n05.msc.itso.ibm.com and sp4n10.msc.itso.ibm.com are group leaders (data managers).

Installation of the PTPE performance hierarchy using the definitions shown in Figure 99 on page 238 was done with `ptpehier`. The procedure we used is shown in Figure 100.

```
# /usr/lpp/ptpe/bin/ptpehier -i -c sp4n01 < /etc/ptpe_hierarchy
ptpehier: Monitoring hierarchy successfully created.
```

Figure 100. Installing a PTPE Performance Hierarchy

Note

The node sp4n01 is designated as the central coordinator using the ptpehier -c <central coordinator> option.

With the PTPE performance monitoring hierarchy installed, we initialized PTPE for data collection using `ptpectrl -i`. Figure 101 shows that PTPE is now set up to monitor the performance of our complex.

```
# ptpectrl -i
------------------------------------------------------------
ptpectrl: Beginning setup for performance information collection.
ptpectrl: Reply from the Central Coordinator expected within 255 seconds. OK.
ptpectrl: Setup for collecting performance information succeeded.
------------------------------------------------------------
ptpectrl: Command completed.
------------------------------------------------------------
```

Figure 101. Initializing a PTPE Performance Hierarchy

11.8.2 Using PTPE to Monitor an RS/6000 SP Cluster

PTPE is an extension to the performance toolbox (PTX/6000) for AIX.

To view or monitor the performance of an RS/6000 SP complex with PTPE, the runtime monitor from PTX/6000 is required. A brief introduction to PTX/6000 can be found in 11.7, “Performance Toolbox for AIX (PTX/6000)” on page 222.

PTPE aggregates system statistics only when enabled. This eliminates the overhead of collecting and maintaining these statistics when they are not required.

To enable collection of aggregated statistics, use `ptpectrl` as shown in Figure 102.

```
# ptpectrl -c
------------------------------------------------------------
ptpectrl: Starting collection of performance information.
ptpectrl: Reply from the Central Coordinator expected within 255 seconds. OK.
ptpectrl: Performance information collection successfully started.
------------------------------------------------------------
ptpectrl: Resetting statistic controls for collection.
ptpectrl: Reply from the Central Coordinator expected within 270 seconds. OK.
ptpectrl: Some systems reported that they were not ready to accept the request.
        Pausing 15 seconds to retry command.
ptpectrl: Attempting the command again.
ptpectrl: Reply from the Central Coordinator expected within 270 seconds. OK.
ptpectrl: Enabling statistics for performance information collection.
ptpectrl: Reply from the Central Coordinator expected within 270 seconds. OK.
ptpectrl: Enabling and restriction of statistics complete.
------------------------------------------------------------
ptpectrl: Command completed.
------------------------------------------------------------
```

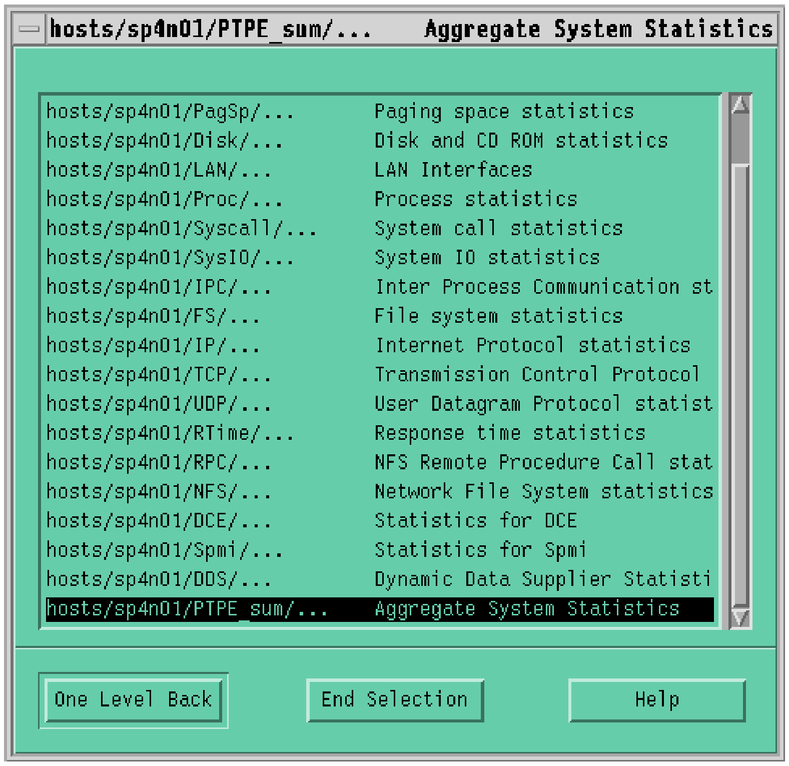
Figure 102. Enabling PTPE Aggregate Data Collection

With the collection of aggregated system statistics enabled, the performance toolbox GUI and libraries have these statistics available when monitoring a data manager or central coordinator.

When monitoring the central coordinator, selecting aggregated system statistics provides aggregated statistics across the complex for all nodes that are participants in the performance monitoring hierarchy.

When monitoring a data manager, selecting aggregated system statistics provides statistics aggregated for all nodes that report to the data manager.

Figure 103 on page 240 is an example of selecting the aggregated statistics when adding values to a PTX/6000 monitoring instrument. In this example, we are selecting aggregated system statistics for the complex from the central coordinator sp4n01.



```
hosts/sp4n01/PTPE_sum/...     Aggregate System Statistics

hosts/sp4n01/PagSp/...        Paging space statistics
hosts/sp4n01/Disk/...         Disk and CD ROM statistics
hosts/sp4n01/LAN/...          LAN Interfaces
hosts/sp4n01/Proc/...         Process statistics
hosts/sp4n01/Syscall/...      System call statistics
hosts/sp4n01/SysIO/...        System IO statistics
hosts/sp4n01/IPC/...          Inter Process Communication st
hosts/sp4n01/FS/...           File system statistics
hosts/sp4n01/IP/...           Internet Protocol statistics
hosts/sp4n01/TCP/...          Transmission Control Protocol
hosts/sp4n01/UDP/...          User Datagram Protocol statist
hosts/sp4n01/RTime/...        Response time statistics
hosts/sp4n01/RPC/...          NFS Remote Procedure Call stat
hosts/sp4n01/NFS/...          Network File System statistics
hosts/sp4n01/DCE/...          Statistics for DCE
hosts/sp4n01/Spmi/...         Statistics for Spmi
hosts/sp4n01/DDS/...          Dynamic Data Supplier Statisti
hosts/sp4n01/PTPE_sum/...     Aggregate System Statistics
```

Figure 103. Selecting PTPE Aggregate System Statistics

When monitoring of the complex or node groups has been completed, the gathering of aggregated system performance data should be disabled. This eliminates the additional overhead the gathering of aggregated system statistics places on nodes participating in the performance monitoring hierarchy.

To disable collection of aggregated statistics, use `ptpectrl` as shown in Figure 104 on page 241.

```
# ptpectrl -s
------------------------------------------------------------
ptpectrl: Reply from the Central Coordinator expected within 270 seconds. OK.
ptpectrl: Performance information collection successfully stopped.
------------------------------------------------------------
ptpectrl: Command completed.
------------------------------------------------------------
```

Figure 104. Disabling PTPE Aggregate Data Collection

For further information about PTPE, refer to *Customizing Performance Toolbox and Performance Toolbox Parallel Extensions for AIX,* SG24-2011, or *IBM Parallel System Support Programs for AIX Performance Monitoring Guide and Reference Version 3 Release 1,* SA22-7353.

Chapter 12. Non-IBM Performance Tools

Besides the set of powerful performance monitoring and tuning tools from IBM, a couple of other mostly free performance tools are also worth mentioning. The following list of additional tools does not claim to be complete, but these are tools we worked with and can recommend.

12.1 The Real Time IBM RS/6000 AIX System Monitor

This tool was developed mainly by Jussi Maki, who works in the Computing Center of the Helsinki University of Technology (HUT) as a systems analyst in Systems Support Division.

When the HUT bought their first RS/6000 machines in 1990, they were looking for a performance monitoring tool that provides constantly updated AIX statistics on a *dumb* screen.

The only tools available at that time were the standard UNIX statistics tools ps, vmstat, iostat, nfstat, netstat, and so on. These raw tools do not give an *at a glance* view, which was needed when monitoring the entire systems utilization.

They decided to develop their own performance monitor. It is a text screen application that periodically updates most of the available system status information. The tool is available at no charge.

How do I get it? It you are interested in getting the source code, visit the authors Web-page at `http://www.csc.fi/~jmaki/` and download the appropriate package. Bear in mind that you will need a C Compiler on your system.

Another solution is to look for a binary version and download it. You will find it on `http://www.bull.de`. On this server, bull provides a lot of freeware packages in installable lpp format.

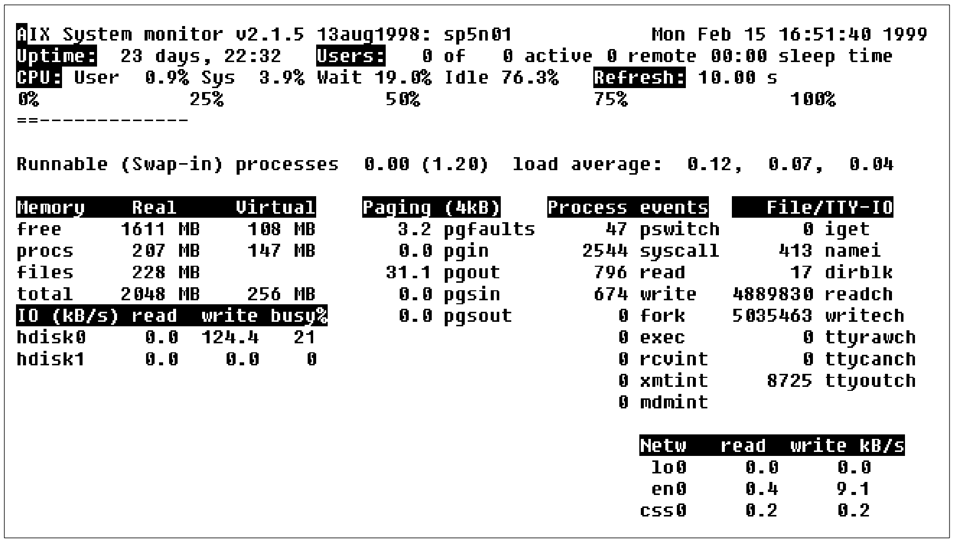
Here is a list of system events that are monitored:

- CPU usage
- Load average (from kernel or by using the loadavgd program)
- Virtual and real memory usage (both process and file pages)
- Paging information
- Process events
- Disk I/O
- TTY I/O

- Network activity
- Top CPU users
- NFS operations
- More detailed disk I/O screen (with the -disk option)
- More detailed information on multiprocessor SMP machines

Brief Tutorial

You can start `monitor` without command line options. The screen (see Figure 105) contains all the vital information about a running system.



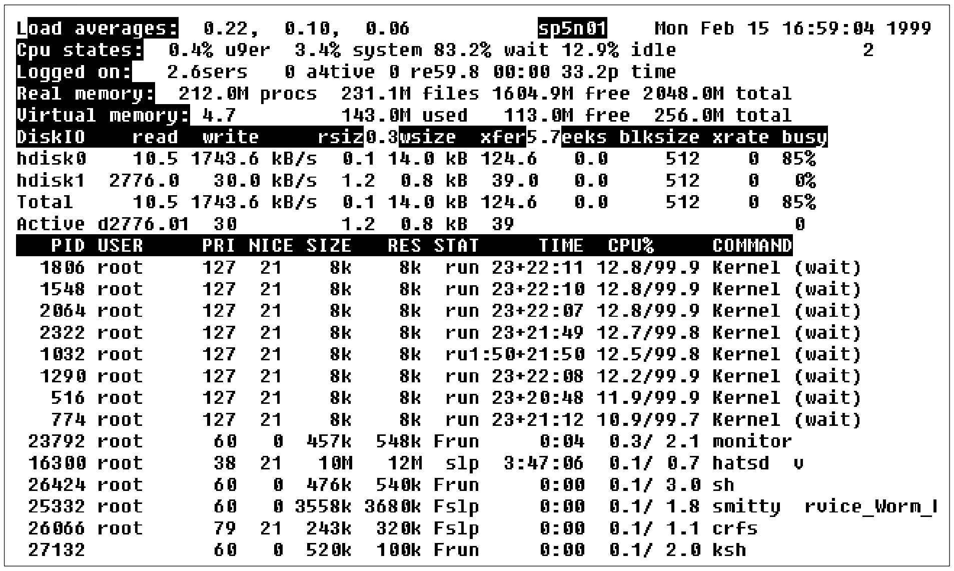
```
AIX System monitor v2.1.5 13aug1998: sp5n01           Mon Feb 15 16:51:40 1999
Uptime:  23 days, 22:32   Users:   0 of   0 active 0 remote 00:00 sleep time
CPU: User  0.9% Sys  3.9% Wait 19.0% Idle 76.3%   Refresh: 10.00 s
0%           25%              50%              75%              100%
===============

Runnable (Swap-in) processes  0.00 (1.20)  load average:  0.12,  0.07,  0.04

Memory     Real      Virtual    Paging (4kB)    Process events    File/TTY-IO
free      1611 MB    108 MB        3.2 pgfaults       47 pswitch        0 iget
procs      207 MB    147 MB        0.0 pgin         2544 syscall      413 namei
files      228 MB                 31.1 pgout         796 read          17 dirblk
total     2048 MB    256 MB        0.0 pgsin         674 write    4889830 readch
IO (kB/s) read  write busy%        0.0 pgsout          0 fork     5035463 writech
hdisk0      0.0  124.4   21                            0 exec           0 ttyrawch
hdisk1      0.0    0.0    0                            0 rcvint         0 ttycanch
                                                       0 xmtint      8725 ttyoutch
                                                       0 mdmint

                                                        Netw   read  write kB/s
                                                         lo0    0.0     0.0
                                                         en0    0.4     9.1
                                                        css0    0.2     0.2
```

Figure 105. monitor - Without Options

There are more details about Disk, Network, and Top processes on other screens. To get to these, just type the first letter:

- d = disks
- n = network
- t = top processes
- s = SMP CPU details
- a = show more (all) of the previously selected screen
- h = help
- q = QUIT

Figure 106 on page 245 shows the results of using the monitor with the disk and top options.



```
Load averages:  0.22,  0.10,  0.06          sp5n01    Mon Feb 15 16:59:04 1999
Cpu states:  0.4% u9er  3.4% system 83.2% wait 12.9% idle                 2
Logged on:   2.6sers   0 a4tive 0 re59.8 00:00 33.2p time
Real memory:  212.0M procs  231.1M files 1604.9M free 2048.0M total
Virtual memory: 4.7         143.0M used   113.0M free  256.0M total
DiskIO    read  write      rsiz0.3wsize  xfer5.7eeks blksize xrate busy
hdisk0    10.5 1743.6 kB/s  0.1 14.0 kB 124.6   0.0     512    0  85%
hdisk1  2776.0   30.0 kB/s  1.2  0.8 kB  39.0   0.0     512    0   0%
Total     10.5 1743.6 kB/s  0.1 14.0 kB 124.6   0.0     512    0  85%
Active d2776.01  30         1.2  0.8 kB  39                          0
   PID USER      PRI NICE SIZE   RES STAT      TIME  CPU%     COMMAND
  1806 root      127  21    8k    8k  run 23+22:11 12.8/99.9 Kernel (wait)
  1548 root      127  21    8k    8k  run 23+22:10 12.8/99.9 Kernel (wait)
  2064 root      127  21    8k    8k  run 23+22:07 12.8/99.9 Kernel (wait)
  2322 root      127  21    8k    8k  run 23+21:49 12.7/99.8 Kernel (wait)
  1032 root      127  21    8k    8k  ru1:50+21:50 12.5/99.8 Kernel (wait)
  1290 root      127  21    8k    8k  run 23+22:08 12.2/99.9 Kernel (wait)
   516 root      127  21    8k    8k  run 23+20:48 11.9/99.9 Kernel (wait)
   774 root      127  21    8k    8k  run 23+21:12 10.9/99.7 Kernel (wait)
 23792 root       60   0  457k  548k Frun     0:04  0.3/ 2.1 monitor
 16300 root       38  21   10M   12M  slp  3:47:06  0.1/ 0.7 hatsd  v
 26424 root       60   0  476k  540k Frun     0:00  0.1/ 3.0 sh
 25332 root       60   0 3558k 3680k Fslp     0:00  0.1/ 1.8 smitty  rvice_Worm_I
 26066 root       79  21  243k  320k Fslp     0:00  0.1/ 1.1 crfs
 27132            60   0  520k  100k Frun     0:00  0.1/ 2.0 ksh
```

Figure 106. monitor - Disk and Top Statistics

It is beyond the scope of this book to describe all the options of monitor. For more detailed information, look at the manpages after installing the monitor.

12.2 POWER2 Hardware Performance Monitoring

Besides the more universal performance tool described in the preceding section, the same author offers a free Hardware Performance Monitoring (HPM) tool especially for IBM POWER2 CPUs. This package consists of a library, a data collection daemon, kernel extensions, and a few utilities. These tools can be used only with systems that have IBM POWER2 CPUs (for example, RS/6000 model 590, model 390, or some models of IBM SP).

This tools package is available as free software. Send an e-mail to Jussi.Maki@csc.fi to get the package. Due to some confidentiality in POWER2 CPU internals, the distribution will be in source code form after IBM's approval.

HPM tools are especially useful when tuning and benchmarking floating-point operation-intensive applications. The tools can also be used to monitor CPU utilization on different parts (for example, fixed-point units, floating-point units, and cache operations). Actually, the POWER2 CPU has dozens of measurable events. These tools address only a subset of these concentrating on fixed-point and floating-point operations and cache usage.

There are some limitations in multiuser environments: measurement is done for the whole system (utilization in user and kernel parts can be separated). Single application measurements can be done when there are no other users using the CPU. Real multiuser support would require support for kernels by IBM, which is not available at the moment. The system, however, provides multiple monitoring sessions and a single-user measurement session.

The data collection consists of a single-user space server, rs2hpmProvider, which accesses the POWER2 HPM registers by using a preloaded rs2hpm-kex kernel extension. The server process collects the 32-bit counters in regular intervals and stores them in 64-bit format; so, the data will not overflow for measurements longer than a few minutes. The server also enables access control to data instead of several users at the same time accessing the actual hardware counters and thus obtaining possible garbage information.

Figure 107 shows the appearance of rs2mon.

```
IBM POWER2 HARDWARE PERFORMANCE MONITOR
Host: cactus-1                   Date: 95/10/18 Time: 18:15:31

USER   MODE   CACHE              SYSTEM MODE    SYSTEM CACHE
CPU%    97.7  DataM  9776.1k     CPU%    2.3    DataM    39.7k
MIPS   124.0  DataS   268.7k     MIPS    0.7    DataS    14.6k
MFLOPS 184.1  DataRL  651.9k     DMAr    0.0    DataRL   38.8k
Fadds   92.1  InstRL    6.0k     DMAw    0.0    InstRL  494.4k
Fmuls    0.0  TLBmis    4.4k                    TLBmis    2.7k
Fmas    92.1

CPU Units usage SCU usage       CPU Units usage  SCU usage
FXU0    47.3M   Mreads 660.7k    FXU0      0.4M  Mreads   0.0k
FXU1    28.6M   Move   268.7k    FXU1      0.2M  Move    14.6k
FPU0    46.2M                    FPU0      0.0M
FPU1    45.8M                    FPU1      0.0M
ICU     48.1M                    ICU       0.1M
```

Figure 107. rs2mon Output

The server provides the data via the TCP/IP interface using a simple request and acknowledgment protocol.

Part of the toolkit is the `sp2flops` command, which relies on the client-server architecture of the package. sp2flops shows hardware performance data for the SP machines using POWER2 CPU HPM information.

Our example in Figure 108 shows a subset of SP nodes (nodes 10-16).

```
% sp2flops
SP2FLOPS        1995-10-19 21:33        Data cache operations
Nodename        US%  SY%   MIPS MFLOPS   Store  Reload  Misses
cactus-10       99%   1%   34.9   70.2  2883.8k 14667.9k  307.6k
cactus-11       98%   2%   34.8   28.6  1815.3k  7971.8k 1661.4k
cactus-12       98%   2%   34.8   28.6  1828.9k  8041.1k 1657.8k
cactus-13       99%   1%   56.7   72.1  1339.0k 10350.7k 1055.3k
cactus-14       98%   2%   34.7   28.6  1820.0k  8018.8k 1656.9k
cactus-15       97%   3%   29.2   61.0  2402.4k 12080.6k  257.2k
cactus-16       98%   2%   29.9   62.6  2470.8k 12279.8k  223.5k
Total           98%   2%  255.0  351.8
```

Figure 108. sp2flops Output

For more information, refer to the author’s Web page.

12.3 NetPerf

Netperf is a benchmark in the public domain that can be used to measure various aspects of networking performance. The primary focus of this package is on bulk data transfer and request/response performance with CPU utilization using either TCP or UDP and the Berkeley Sockets interface. Netperf is also a good testing tool for communication and network software. It was designed around the basic client-server with point-to-point model. It was originally designed by Hewlett-Packard (HP) and is now maintained and informally supported by the IND Networking Performance Team. It is *not* supported by any of the normal Hewlett-Packard support channels.

Netperf is distributed in source form. This allows installation on a variety of systems. There are two ways to install netperf. The first runs the netperf server program, netserver, as a child of inetd, which requires that the installer of netperf be able to edit the files /etc/services and /etc/inetd.conf (or their equivalent).

The second is to run netserver as a standalone daemon. This method does not require edit capabilities on /etc/services and /etc/inetd.conf, but does mean that you must remember to run the netserver program explicitly.

Netperf is designed around the basic client-server model. There are two executables: netperf and netserver. Generally, you will only execute the netperf program. The netserver program will be invoked by the other system's inetd.

When you execute netperf, the first thing that happens is the establishment of a control connection to the remote system. This connection is used to pass test configuration information and results to and from the remote system. Regardless of the type of test being run, the control connection is a TCP connection using BSD sockets.

Once the control connection is up and the configuration information has been passed, a separate connection is opened for the measurement itself using the APIs and protocols appropriate for the test. The test is performed, and the results are displayed.

Netperf places no traffic on the control connection while a test is in progress. Certain TCP options, such as SO_KEEPALIVE, if set as your system's default, may put packets out on the control connection.

The most common use of netperf is for measuring bulk data transfer performance. This is also referred to as *stream* or *unidirectional stream* performance. Essentially, these tests measure how fast one system can send data to another and/or how fast that other system can receive it.

12.3.1 TCP Stream Performance

The TCP stream performance test is the default test type for the netperf program. The simplest test is performed by entering the command:

```
netperf -H remotehost
```

This performs a 10 second test between the local system and the system identified by remotehost. The socket buffers on either end are sized according to the systems' defaults and all TCP options (for example TCP_NODELAY) are at their default settings.

To assist in measuring TCP stream performance, two script files are provided with the netperf distribution: tcp_stream_script and tcp_range_script. Tcp_stream_script invokes netperf based on the setting of script variables controlling socket and send sizes. Tcp_range_script performs a similar set of tests, the difference being that where tcp_stream_script tests specific datapoints, tcp_range_script performs tests at points within a specified range.

If you would like to perform tests other than those done by the scripts, you can invoke netperf manually. Some of the options you will likely want to experiment with are:

- -s sizespec, which will set the local send and receive socket buffer sizes to the value(s) specified. Default: system default socket buffer sizes.
- -S sizespec, which behaves just like -s but for the remote system.
- -m value, which sets the local send size to value bytes. Default: local socket buffer size.
- -M value, which behaves like -m setting the receive size for the remote system. Default: remote receive socket buffer size.
- -l value, which sets the test length to value seconds when value is > 0 and to |value| bytes when value is < 0.
- -D, which sets the TCP_NODELAY option to true on both systems.

This is not a complete list of options that can affect TCP stream performance, but it does cover the options that are used most often.

12.3.2 UDP Stream Performance

A UDP stream performance test is very similar to a TCP stream test. One difference is that the send size cannot be larger than the smaller of the local and remote socket buffer sizes. What this means is that you must make certain that when you specify the -m option, you use a value that is less than or equal to the socket buffer sizes (-s and -S). Also, since the UDP stream test is not the default test, the -t testname option must be specified with testname set to UDP_STREAM. So, a simple UDP stream test command might look something like this:

```
netperf -H remotehost -t UDP_STREAM -- -m 1024
```

The script udp_stream_script is provided to perform various UDP stream performance tests. As with TCP stream performance, you can use the script provided or perform tests yourself to get data points not covered by the script.

Note

UDP is an unreliable protocol. It is important that you examine the results carefully because the reported send rate can be much higher than the actual receive rate. Great care should be taken when reporting UDP_STREAM test results to make sure they are not misleading. For example, one should always report both send and receive rates together for a UDP_STREAM test. If you only report a single number, you should report the receive rate.

Request/response performance is the second area that can be investigated with netperf. Generally speaking, netperf request/response performance is quoted as *transactions/s* for a given request and response size. A transaction is defined as the exchange of a single request and a single response. From a transaction rate, one can infer one-way and round-trip average latency.

12.3.3 TCP Request/Response Performance

The TCP request/response test can be invoked with netperf through the use of the -t option with an argument of TCP_RR. So, a *default* request/response command would look something like this:

```
netperf -H remotehost -t TCP_RR
```

and use the system default socket buffer sizes, a default request size of 1 byte, and a default response size of 1 byte.

As with the stream performance tests, a script is available to assist you in generating TCP request/response performance numbers: tcp_rr_script. However, if you should need to generate numbers at points of your own choosing, these command line options will be of use:

- -r sizespec sets the request and/or response sizes based on sizespec.
- -l value sets the test duration based on value. For value > 0, test duration will be value seconds. Otherwise, test duration will be |value| transactions.
- -s sizespec sets the local send and receive socket buffer sizes to the value(s) specified (default: system default socket buffer sizes).
- -S sizespec, which behaves just like -s but for the remote system.
- -D sets the TCP_NODELAY option to true on both systems.

The request and response sizes will be the buffer sizes posted to send and receive. The -m and -M options are not meaningful for a TCP_RR test. Since TCP is a stream protocol and not a message protocol, it is necessary to loop on receives until the entire message is delivered. The buffer pointer passed to the first receive for an individual transaction will be aligned and offset as requested by the user. It will be incremented by the number of bytes received each time until the entire request/response is received. The buffer pointer will be realigned and offset for the next transaction.

12.3.4 UDP Request/Response Performance

UDP request/response performance works just like TCP request/response performance. All the options available for the latter are present except for the -D option; TCP_NODELAY has no meaning for a UDP test. To invoke a UDP request/response test, use an argument of UDP_RR with the -t option to produce a command something like this:

```
netperf -H remotehost -t UDP_RR
```

Again, a script is provided which will generate results for some of the more common data points: udp_rr_script.

These are only some examples of the use of netperf. For more information look at:

```
http://www.netperf.org
```

Here, you will find pointers to the latest document and the latest source code.

12.4 ttcp Program

The public domain ttcp program was developed by T.C. Slattery, USN. The code was modified several times by various authors The latest known version is 3.8, available from ftp servers at CERN.

The ttcp program is easy to use for TCP performance measurements. It generates and transfers synthetic data. Also, you can feed an input stream to ttcp that will then be transferred.

There is no good documentation available for this package; you have to look at the source code.

12.5 Other Commercial Performance Monitor Sources

In addition to IBM products, other commercial performance monitoring applications are available from various vendors like BGS Systems (BEST/1), CIS Inc., and Landmark Systems Corporation.

Appendix A. Performance Problem Checklist

1. Check all nodes and the Control Workstation for default AIX network tunables:

   no -a

2. Check Control Workstation network adapters for transmit queue overflow errors particularly the adapters to the SP Ethernet:

- entstat
- tokstat
- fddistat
- atmstat

3. Check all node Ethernet adapters for transmit queue overflow errors. Pay particular attention to rack SP Ethernet gateway nodes:

   dsh -a "entstat en0 | grep Overflow"

   Similarly, for SP Ethernet gateway nodes:

   dsh -a "entstat en1 | grep Overflow"

4. Check to see that the ARP cache is set correctly for systems greater than 128 nodes:

   arp -a | wc -l.

5. Check vdidl2/3 for failed counts on the switch send pool:

   /usr/lpp/ssp/css/vdidl3 -i

6. Check nodes using the netstat -i output for unexpected or unbalanced traffic flow.
7. For NFS, check for client timeout and retry counts. These indicate an NFS server that has too few nfsd daemons and/or too small an nfs_socketsize:

   nfsstat -c

8. Check ps gvc for runaway processes:
   - Look for processes causing lots of pagein counts.
   - Look for processes using up lots of memory
9. Check vmstat for paging:
   - Check for page scan rates in the *sr* column.
   - Check for page freeing rates in the *fr* column.

Appendix B. Hardware Details

In this appendix, we give an overview of the nodes available at the time the book was written plus a node selection criteria scheme that may help you find the right node for your application.

B.1 Node Types

The SP processor nodes are built from and architecturally equivalent to specific RS/6000 server models. The 160 MHz thin and the 135 MHz wide nodes use P2SC (POWER2 Super Chip) while the high node is supplied by 604e PowerPC processors. The 332 MHz thin/wide nodes use 604+LX PowerPC, while the 200 MHz thin/wide SMP nodes use POWER3© processors. The S70 node uses 64-bit PowerPC RS64, and S7A uses 64-bit PowerPC RS64 II.

Table 28. SP Nodes Overview

Node Type	Processor	Data Cache	MCA/PCI Slots Available		Memory Bus Bandwidth
			no Switch	using Switch	
Thin Node 160 MHz (397)	P2SC	128 KB	4	3	256-bit
Wide Node 135 MHz (595)	P2SC	128 BK	7	6	256-bit
High Node 200 MHz (J50)	PowerPC [1]	32K + 2MB L2 [5]	14	13	256-bit
Thin Node 332 MHz (H50)	PowerPC [2]	32K + 256KB L2 [5]	2 [6]	2 [7]	128-bit
Wide Node 332 MHz (H50)	PowerPC [2]	32K + 256KB L2 [5]	10 [6]	10 [7]	128-bit
S70 Node 125 MHz	PowerPC [3]	64K + 4MB L2 [5]	11 [6]	8 [8]	Dual 512-bit
S7A Node 262 MHz	PowerPC [4]	64K + 8MB L2 [5]	11 [6]	8 [8]	Dual 512-bit
Thin Node 200 MHz	POWER3 [9]	64K + 4MB L2 [5]	2 [6]	2 [7]	128-bit
Wide Node 200 MHz	POWER3 [9]	64K + 4MB L2 [5]	10 [6]	10 [7]	128-bit

[1] 2/4/6/8-way PowerPC 604e.
[2] 2/4-way PowerPC 604+LX.
[3] 4-way PowerPC RS64 64-bit.
[4] 4-way PowerPC RS64 II 64-bit.
[5] This data cache is for each pair of processors.

[6] These slots are PCIs.
[7] SP Switch MX Adapter does not use an I/O slot.
[8] SP System Attachment Adapter requires 3 PCI slots.
[9] 1/2-way 64-bit processor.

Characteristics of these nodes vary by:

- The bandwidth of the internal bus.
- The number of CPUs (the high nodes are SMP systems with 2, 4, 6, or 8 processors).
- The maximum amount of memory.
- The number of available microchannels or PCI adapter slots.
- The amount of memory bandwidth (one word = 32 bits).

The memory bandwidth is important when considering system performance. The memory bandwidth word size specifies how many words of data can be moved between memory and data cache per CPU cycle.

B.2 Roles of Nodes

There are a variety of nodes to choose from when you configure an SP system. The following is a short overview that gives some guidance for your node selection.

Thin Node
Thin nodes are suitable for a wide range of applications including all commercial applications and most scientific and technical applications.

Wide Node
Wide nodes in some cases demonstrate superior performance over thin nodes even though they share the same type and number of processors. This is due to the existence in the wide node of a second controller that attaches eight additional slots to the internal bus. For some I/O-related performance, this could be significant. For example, with SSA disks, the data rate performance increases from about 40 MBps on a thin PCI node to more like 70 MBps on a wide PCI node. This alone might be reason enough for selecting wide nodes rather than thin nodes. The incremental cost in using wide nodes rather than thin nodes may well be a good investment if balanced performance is of importance. A specific application example where the extra performance is likely to be observed when using a PCI wide node would be when using the PCI node as a disk server node for a GPFS file system. Up to double the bandwidth could be achieved with a wide node in this case.

High Node
High nodes are more suited for multithreaded applications such as commercial database processing.

S70/S7A External Node
The S70/S7A External Server is a PowerPC RS/6000 model capable of 32- or 64-bit processing. It is best suited as a powerful database server. Both nodes can have 4-, 8-, or 12-way processors. The S70 has a 125 MHz PowerPC RS64 processor. The S7A has a PowerPC RS64II processor with 262 MHz.

Router Node
Router nodes are dependent nodes that extend the RS/6000 SP system's capabilities. They are not housed in the frame but are dependent on the switch existing in the environment. An RS/6000 SP Switch Router is a high-performance I/O gateway for the RS/6000 SP system and provides the fastest available means of communication between the RS/6000 SP system and the outside world or among multiple RS/6000 SP systems. The Ascend GRF switched IP router can be connected to the SP switch via the SP Router Adapter. The Switch Router Adapter provides a high-performance 100 MBps full duplex interface between the SP Switch and the Ascend GRF.

B.3 Communication Paths

Two communication paths between the nodes and the Control Workstation (Ethernet network) and between the frame and the Control Workstation are mandatory for an SP system. The switch network is optional.

RS232 Hardware Monitoring Line
The mandatory RS232 hardware monitoring line connects the CWS to each RS/6000 SP frame used primarily for node and frame hardware monitoring.

Ethernet
One of the prerequisites of the RS/6000 SP is an internal bnc or 10BaseT network. The purpose of this mandatory network is to install the nodes' operating systems and PSSP software as well as to diagnose and maintain the RS/6000 SP complex through the PSSP software.

B.4 System Partitioning

System partitioning within the RS/6000 SP allows you to divide your system into logically separate systems. The concept of system partitioning is very similar to the Virtual Machine in the mainframe environment, which supports different machine *images* running in parallel; you can have a production image and test image within the same machine. The RS/6000 SP provides completely isolated environments within the complex.

Although RS/6000 SP system partitioning has been used as a migration tool to isolate test environments, this is not its primary purpose. Running various versions of AIX and PSSP is possible without having to partition.

Now that you can differentiate between production and testing within a complex, you can simulate a production environment and take all necessary steps to tune your system before migrating to the true production environment.

B.5 Node Selection Process

There are different criteria for choosing the right node for your application. One criterion may be capacity: how many adapters, how many internal disks or how much memory will fit into it.

Capacity
When selecting a node, you may choose to use the methodology shown in the flowchart in Figure 109. Look at the required capacity for adapters, disks, and memory; this will help you decide whether a thin node has sufficient capacity. As you consider this area, take particular care to include all adapters that will be required both now and in the near future.

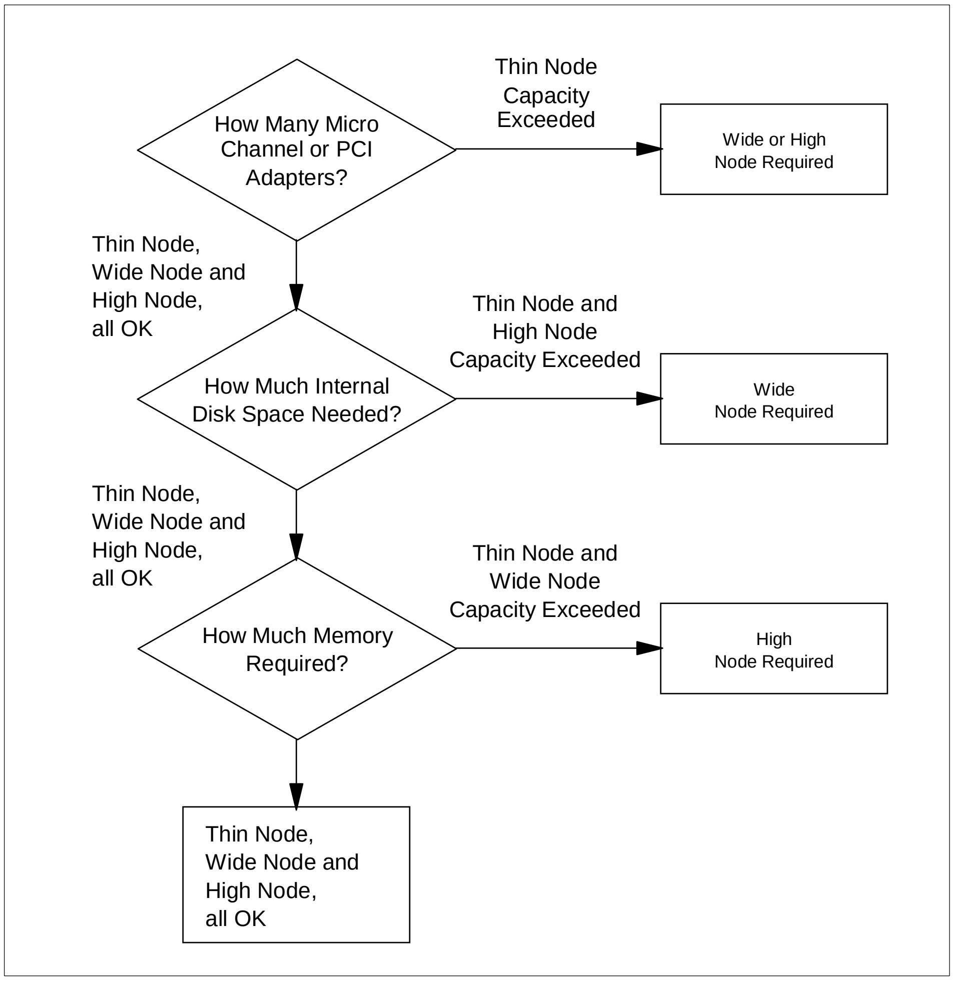

How Many Micro Channel or PCI Adapters?
Thin Node Capacity Exceeded
Wide or High Node Required
Thin Node, Wide Node and High Node, all OK
How Much Internal Disk Space Needed?
Thin Node and High Node Capacity Exceeded
Wide Node Required
Thin Node, Wide Node and High Node, all OK
How Much Memory Required?
Thin Node and Wide Node Capacity Exceeded
High Node Required
Thin Node, Wide Node and High Node, all OK

Figure 109. RS/6000 SP Node Selection Based on Capacity

Some of the selections that need to be made are quite straightforward decisions and are based on concrete factors, such as the number of adapters needed in this node or the amount of memory required.

Performance
The flow chart in Figure 110 on page 260 can be used to help make the right choice of nodes to select from a performance perspective. It assumes you have already made a decision on the basis of capacity as discussed earlier.

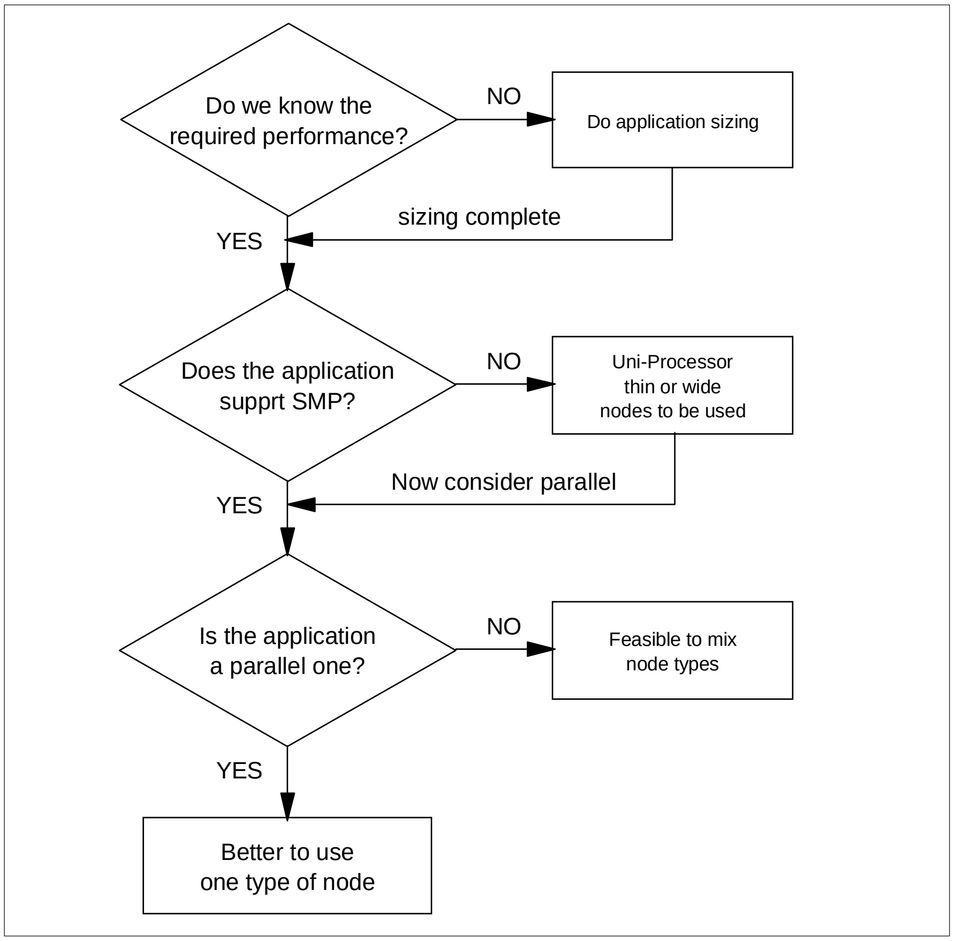

Do we know the required performance?
NO
Do application sizing
sizing complete
YES
Does the application supprt SMP?
NO
Uni-Processor thin or wide nodes to be used
Now consider parallel
YES
Is the application a parallel one?
NO
Feasible to mix node types
YES
Better to use one type of node

Figure 110. RS/6000 SP Node Selection Based on Performance

Performance Measurements

It is important to select the correct nodes for the applications that we wish to implement and that exploit the most appropriate architecture: serial or parallel, uniproccesors or SMP processors. Make sure that you have a cost-effective solution that also allows easy upgrades and a growth path as required.

It is important not to be swayed in these considerations by any price/performance considerations that force you to make compromises with regard to the ideal nodes for a specific purpose. The good news is that price/performance for all RS/6000 SP nodes, be they uniprocessor or high nodes, is similar for the current selection of nodes.

Appendix C. No Command Man Page

Here is an excerpt from the `no` command man page for AIX 4.3.2:

Purpose

Configures network attributes.

Syntax

```
no {-a | -d Attribute | -o Attribute [ =NewValue ] }
```

Description

Use the `no` command to configure network attributes. This command sets or displays current network attributes in the kernel. It only operates on the currently running kernel. It must be run again after each startup or after the network has been configured. Whether the command sets or displays an attribute is determined by the accompanying flag. The `-o` flag performs both actions. It can either display the value or set a new value for an attribute. For more information on how the network attributes interact with each other, refer to the *AIX System Management Guide: Communications and Networks,* GC23-2487*.*

Note

Be careful when you use this command. The `no` command performs no range checking and therefore accepts all values for the variables. If used incorrectly, the `no` command can cause your system to become inoperable.

Some network attributes are runtime attributes that can be changed at any time. Others are loadtime attributes that must be set before the netinet kernel extension is loaded and must be placed near the rop of /etc/rc.net. If your system uses a Berkeley style network configuration, set the attributes near the top of /etc/rc.bsdnet.

Flags:

-a	Prints a list of all configurable attributes and their current values.
-d	Sets an attribute back to its default value.
-o	[=NewValue] Displays the value of an attribute if NewValue is not specified; otherwise, it sets an attribute to NewValue.

Note

When using the `-o` flag, do not enter space characters before or after the equal sign. If you do, the command will fail.

C.1 Network Attributes

You can set the following attributes:

arpqsize

This specifies the maximum number of packets to queue while waiting for ARP responses. Default value is 1. This attribute is supported by Ethernet, 802.3, Token Ring, and FDDI interfaces. The arpqsize value is increased to a minimum of 5 when path MTU discovery is enabled. The value will not automatically decrease if path MTU discovery is subsequently disabled. This attribute applies to AIX Versions 4.1.5, 4.2.1, and later. arpqsize is a runtime attribute.

arptab_bsiz

This specifies Address Resolution Protocol (ARP) table bucket size. The default value is 7. arptab_bsiz is a loadtime attribute.

arptab_nb

This specifies the number of ARP table buckets. The default value is 25. arptab_nb is a loadtime attribute.

arpt_killc

This specifies the time in minutes before a complete ARP entry will be deleted. The default value is 20 minutes. arpt_killc is a runtime attribute.

bcastping

This allows response to ICMP echo packets to the broadcast address. A value of 0 turns it off, while a value of 1 turns it on. Default is 0. bcastping is a runtime attribute.

clean_partial_conns

This specifies whether or not to avoid SYN attacks. If true, randomly remove partial connections to make room for new non-attack connections. This is a runtime attribute. The default is 0, that is, off.

delayack

This delays ACKs for certain TCP packets and attempts to piggyback them with the next packet sent instead. This will only be performed for connections whose destination port is specified in the list of the delayackports attribute. This can be used to increase performance when communicating with an HTTP server. It is available only in AIX 4.3.2 and beyond. The attribute can have one of four values:

0 - No delays; normal operation

1 - Delay the ACK for the server's SYN

2 - Delay the ACK for the server's FIN

3 - Delay both the ACKs for the SYN and FIN

delayackports

This specifies the list of destination ports for which the operation defined by the delayack port option will be performed. It takes a list of up to ten ports separated by commas and enclosed in curly braces, for example, no -o delayackports={80,30080}. To clear the list, set the option to {}. This attribute is available only in AIX 4.3.2 and beyond.

extendednetstats

This enables more extensive statistics for network memory services. The default for this attribute is 1. However, because these extra statistics cause a reduction in system performance, extendednetstats is set to 0, for off, in /etc/rc.net. If these statistics are desired, it is recommended that the code in /etc/rc.net that sets extendednetstats to 0 be commented out. This attribute is available only in AIX 4.3.2 and beyond.

directed_broadcast

This specifies whether or not to allow a directed broadcast to a gateway. The value of 1 allows packets to be directed to a gateway to be broadcast on a network on the other side of the gateway. This is a runtime attribute.

icmpaddressmask

This specifies whether the system responds to an ICMP address mask request. If the default value 0 is set, the network ignores any ICMP address mask request that it receives. This is a runtime attribute.

ie5_old_multicast_mapping

This specifies that IP multicasts on Token Ring should be mapped to the broadcast address rather than a functional address when value 1 is used. The default value is 0. This is a runtime attribute.

ifsize

This specifies the maximum number of network interface structures per interface. The default value is 8. In AIX 4.3.2 and later, if the system detects at boot time that more adapters of a type are present than would be allowed by the current value of ifsize, it will automatically increase the value to support the number of adapters present. ifsize is a loadtime attribute.

inet_stack_size

This lets you configure the inet interrupt stack table size. This is needed if you are running with an unoptimized debug kernel and/or netinet. It must be set in rc.net; changing it on the fly has no effect. This is different from the pin more stack code because this is on interrupt. The pin more stack code is not configurable. inet_stack_size is specified in KB (the default is 16 KB).

ipforwarding

This specifies whether the kernel should forward packets. The default value of 0 prevents forwarding of IP packets when they are not for the local system. A value of 1 enables forwarding. This is a runtime attribute.

ipfragttl

This specifies the time to live for IP fragments. The default value is 60 half seconds. This is a runtime attribute.

ipqmaxlen

This specifies the number of received packets that can be queued on the IP protocol input queue. This is a loadtime attribute.

ipignoreredirects

This specifies whether to process redirects that are received. The default value of 0 processes redirects as usual. A value of 1 ignores redirects. This option only applies to AIX Version 4.2.1 or later. This is a runtime attribute.

ipsendredirects

This specifies whether the kernel should send redirect signals. The default value of 1 sends redirects. A value of 0 does not send redirects. This is a runtime attribute.

ipsrcrouteforward

This specifies whether the system forwards source-routed packets. The default value of 1 allows the forwarding of source-routed packets. A value of 0 causes all source-routed packets that are not at their destinations to be discarded. This attribute only applies to AIX Version 4.2.1 or later.

ipsrcrouterecv

This specifies whether the system accepts source-routed packets. The default value of 0 causes all source-routed packets destined for this system to be discarded. A value of 1 allows source-routed packets to be received. This attribute only applies to AIX Version 4.2.1 or later.

ipsrcroutesend

This specifies whether applications can send source-routed packets. The default value of 1 allows source-routed packets to be sent. A value of 0 causes setsockopt() to return an error if an application attempts to set the source-routing option and removes any source-routing options from outgoing packets. This attribute only applies to AIX Version 4.2.1 or later.

ip6_defttl

This specifies the default hop count that is used for IPv6 packets if no other hop count is specified.

ip6forwarding

This specifies whether the kernel should forward ipv6 packets. The default value of 0 prevents forwarding of ipv6 packets when they are not for the local system. A value of 1 enables forwarding. This is a runtime attribute.

ip6_prune

This specifies how often to check the IPv6 routing table for expired routes. The default is 2 seconds.

ip6srcrouteforward

This specifies whether the system forwards source-routed IPv6 packets. The default value of 1 allows the forwarding of source-routed packets. A value of 0 causes all source-routed packets that are not at their destinations to be discarded.

maxttl

This specifies the time to live for RIP packets. The default is 255 seconds. This is a runtime attribute.

multi_homed

This specifies the level of multihomed ipv6 host support.

0 indicates the original functionality in AIX 4.3.0. 1 indicates that link local addresses will be resolved by querying each interface for the link local address. 2 indicates that link local addresses will only be examined for the interface defined by main_if6. 3 indicates that link local addresses will only be examined for the interface defined by main_if6 and site local addresses will only be routed for the main_site6 interface.

main_if6

This specifies the interface to use for link local addresses. This is only used by autoconf6 to set up initial routes.

main_site6

This specifies the interface to use for site local address routing. This is only used if multi_homed is set to 3.

maxnip6q

This specifies the maximum number of ipv6 packet reassembly queues. The default is 20.

nbc_limit

This specifies the total maximum amount of memory that can be used for the Network Buffer Cache. This attribute is in number of KBytes. The default value is derived from thewall. When the cache grows to this limit, the least-used cache objects are flushed out of cache to make room for the new ones. This attribute only applies to AIX version 4.3.2 or later.

nbc_max_cache

This specifies the maximum size of the cache object allowed in the Network Buffer Cache. This attribute is in number of bytes; the default is 131072 (128K) bytes. A data object bigger than this size is not put in the NBC. This attribute only applies to AIX version 4.3.2 or later.

nbc_min_cache

This specifies the minimum size of the cache object allowed in the Network Buffer Cache. This attribute is in number of bytes; the default is 1 byte. A data object smaller than this size is not put in the NBC. This attribute only applies to AIX version 4.3.2 or later.

ndpqsize

This specifies the number of packets to hold waiting on completion of a Neighbor Discovery Protocol (NDP) entry. The default is 50 packets.

ndpt_down

This specifies the time, in half seconds, to hold down an NDP entry. The default value is 3 units, or 1.5 seconds.

ndpt_keep

This specifies the time, in half seconds, to keep an NDP entry. The default value is 120 or 60 seconds.

ndpt_probe

This specifies the time, in half seconds, to delay before sending the first NDP probe. The default value is 5 units, or 2.5 seconds.

ndpt_reachable

Specifies the time, in half seconds, to test whether an NDP entry is still valid. The default is 30, which means 15 seconds.

ndpt_retrans

This specifies the time in half seconds to wait before retransmitting an NDP request. The default is 1, which means half a second.

ndpt_umaxtries

This specifies the maximum number of Unicast NDP packets to send. The default value is 3.

ndpt_mmaxtries

This specifies the maximum number of Mulitcast NDP packets to send. The default value is 3.

net_malloc_police

This specifies the size of the net_malloc/net_free trace buffer. If the value of this variable is non-zero, all net_malloc and net_free variables are traced in a kernel buffer and by system trace hook HKWD_NET_MALLOC. Additional error checking is enabled. This includes checks for freeing a free buffer, alignment, and buffer overwrite. The default value is zero (policing off). Values of net_malloc_police larger than 1024 allocate that many items in the kernel buffer for tracing. This is a runtime attribute.

nonlocsrcroute

This tells the Internet Protocol that strictly source-routed packets may be addressed to hosts outside the local network. A default value of 0 disallows addressing to outside hosts. A value of 1 allows packets to be addressed to outside hosts. Loosely source-routed packets are not affected by this attribute. This is a runtime attribute.

pmtu_default_age

This specifies the default time (in minutes) before the path MTU value for UDP paths is checked for a lower value. A value of 0 allows no aging. The default value is 10 minutes. The pmtu_default_age value can be overridden by UDP applications. This attribute only applies to AIX Version 4.2.1 or later and is a runtime attribute.

pmtu_rediscover_interval

This specifies the default time (in minutes) before the path MTU values for UDP and TCP paths are checked for a higher value. A value of 0 allows no path MTU rediscovery. The default value is 30 minutes. This attribute only applies to AIX Version 4.2.1 or later and is a runtime attribute.

rfc1122addrchk

This performs address validation as specified by RFC1122, *Requirements for Internet Hosts-Communication Layers*. The default value of 0 does not perform address validation. A value of 1 performs address validation. This is a runtime attribute.

rfc1323

This enables TCP enhancements as specified by RFC 1323, *TCP Extensions for High Performance*. The default value of 0 disables the RFC enhancements on a system-wide scale. A value of 1 specifies that all TCP connections will attempt to negotiate the RFC enhancements. The SOCKETS application can override the default behavior on individual TCP connections using the setsockopt subroutine. This is a runtime attribute.

route_expire

This specifies whether the route expires. A value of 0 allows no route expiration, which is the default. Negative values are not allowed for this option. This attribute only applies to AIX Version 4.2.1 or later. This is a runtime attribute.

routerevalidate

This specifies that each connection’s cached route should be revalidated each time a new route is added to the routing table. This ensures that applications that keep the same connection open for long periods of time (for example NFS) use the correct route after routing table changes occur. The default value of 0 does not revalidate the cached routes. Turning this option on may cause some performance degradation. This is a runtime attribute.

rto_length

This specifies the TCP Retransmit Time Out length value used in calculating factors and the maximum retransmits allowable used in TCP data segment retransmits. rto_length is the total number of time segments. The default is 13. This is a loadtime attribute.

rto_limit

This specifies the TCP Retransmit Time Out limit value used in calculating factors and the maximum retransmits allowable used in TCP data segment retransmits. rto_limit is the number of time segments from rto_low to rto_high. The default is 7. This is a loadtime attribute.

rto_low

This specifies the TCP Retransmit Time Out low value used in calculating factors and the maximum retransmits allowable used in TCP data segment retransmits. rto_low is the low factor. The default 1. This is a loadtime attribute.

rto_high

This specifies the TCP Retransmit Time Out high value used in calculating factors and the maximum retransmits allowable used in TCP data segment retransmits. rto_high is the high factor. The default is 64. This is a loadtime attribute.

sb_max

This specifies the maximum buffer size allowed for a socket. The default is 65,536 bytes. This is a runtime attribute.

send_file_duration

This specifies the cache validation duration for all the file objects that system call send_file accessed in the Network Buffer Cache. This attribute is in seconds. The default is 300, which means 5 minutes. 0 means that the cache will be validated for every access. This attribute only applies to AIX version 4.3.2 or later.

site6_index

This specifies the maximum interface number for site local routing.

sockthresh

This specifies the maximum amount of network memory that can be allocated for sockets. When the total amount of memory allocated by the net_malloc subroutine reaches this threshold, the socket and socketpair system calls fail with an error of ENOBUFS. Incoming connection requests are silently discarded. Existing sockets can continue to use additional memory. The sockthresh attribute represents a percentage of the thewall attribute with possible values of 1 to 100 and a default of 85. sockthresh is a runtime attribute and only applies to AIX Version 4.3.1 or later.

somaxconn

This specifies the maximum listen backlog. The default is 1024 bytes. somaxconn is a runtime attribute and only applies to AIX Versions 4.1.5, 4.2 or later.

subnetsarelocal

This determines whether a packet address is on the local network. It is used by the in_localaddress subroutine. The default value of 1 specifies that addresses that match the local network mask are local. If the value is 0, only addresses matching the local subnetwork are local. This is a runtime attribute.

tcp_ephemeral_low

This specifies the smallest port number to allocate for TCP ephemeral ports. The default is 32768. This attribute is available only in AIX 4.3.1 and beyond.

tcp_ephemeral_high

This specifies the largest port number to allocate for TCP ephemeral ports. The default is 65535. This attribute is available only in AIX 4.3.1 and beyond.

tcp_keepidle

This specifies the length of time to keep the connection active measured in half seconds. The default is 14,400 half seconds (7200 seconds or 2 hours). This is a runtime attribute.

tcp_keepinit

This sets the initial timeout value for a tcp connection. This value is defined in half second units, and defaults to 150, which is 75 seconds. It can be changed to any value with the -o flag. This is a runtime attribute.

tcp_keepintvl

This specifies the interval, measured in half seconds, between packets sent to validate the connection. The default is 150 half seconds (75 seconds). This is a runtime attribute.

tcp_mssdflt

This is the default maximum segment size used in communicating with remote networks. For AIX Version 4.2.1 or later, tcp_mssdflt is only used if pth MTU discovery is not enabled or path MTU discovery fails to discover a path MTU. This is a runtime attribute. The default value is 512.

tcp_ndebug

This specifies the number of tcp_debug structures. The default is 100. This is a runtime attribute.

tcp_pmtu_discover

This enables or disables path MTU discovery for TCP applications. A value of 0 disables path MTU discovery for TCP applications, while a value of 1 enables it. The default value is 0. This attribute only applies to AIX Version 4.2.1 or later. This is a runtime attribute.

tcp_recvspace

This specifies the system default socket buffer size for receiving data. This affects the window size used by TCP. Setting the socket buffer size to 16 KB (16,384) improves performance over Standard Ethernet and token-ring networks. The default is a value of 4096; however, a value of 16,384 is set automatically by the rc.net file or the rc.bsdnet file (if Berkeley-style configuration is issued).

Lower bandwidth networks, such as Serial Line Internet Protocol (SLIP), or higher bandwidth networks, such as Serial Optical Link, should have different optimum buffer sizes. The optimum buffer size is the product of the media bandwidth and the average round-trip time of a packet.

The tcp_recvspace attribute must specify a socket buffer size less than or equal to the setting of the sb_max attribute. This is a runtime attribute, but, for daemons started by inetd, the following command needs to be executed:

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

tcp_sendspace

This specifies the system default socket buffer size for sending data. This affects the window size used by TCP. Setting the socket buffer size to 16 KB (16,384) improves performance over Standard Ethernet and token-ring networks. The default is 4096; however, a value of 16,384 is set automatically by the rc.net file or the rc.bsdnet file (if Berkeley-style configuration is issued).

Lower bandwidth networks, such as Serial Line Internet Protocol (SLIP), or higher bandwidth networks, such as Serial Optical Link, should have different optimum buffer sizes. The optimum buffer size is the product of the media bandwidth and the average round-trip time of a packet, that is, optimum_window=bandwidth * average_round_trip_time

The tcp_sendspace attribute must specify a socket buffer size less than or equal to the setting of the sb_max attribute. This is a runtime attribute, but, for daemons started by inetd, the following command needs to be executed:

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

tcp_timewait

The tcp_timewait option is used to configure how long connections are kept in the timewait state. It is given in 15 second intervals, and the default is 1.

tcp_ttl

This specifies the time to live for TCP packets. The default is 60 ticks (100 ticks per minute). This is a runtime attribute.

thewall

This specifies the maximum amount of memory, in kilobytes, that is allocated to the memory pool. In AIX Version 4.2.1 and earlier, the default value is 1/8 of real memory or 65536 (64 MB), whichever is smaller. In AIX Version 4.3.0, the default value is 1/8 of real memory or 131072 (128 MB), whichever is smaller. In AIX Version 4.3.1, the default value is 1/2 of real memory or 131072 (128 MB), whichever is smaller. In AIX Version 4.3.2 and later, the default value is 1/2 of real memory or 1048576 (1 GB), whichever is smaller. thewall is a runtime attribute.

udp_ephemeral_low

This specifies the smallest port number to allocate for UDP ephemeral ports. The default is 32768. This attribute is available only in AIX 4.3.1 and later.

udp_ephemeral_high

This specifies the largest port number to allocate for UDP ephemeral ports. The default is 65535. This attribute is available only in AIX 4.3.1 and later.

udp_pmtu_discover

This enables or disables path MTU discovery for UDP applications. UDP applications must be specifically written to utilize path MTU discovery. A value of 0 (the default) disables the feature, while a value of 1 enables it. This attribute only applies to AIX Version 4.2.1 or later, and is a runtime attribute.

udp_recvspace

This specifies the system default socket buffer size for receiving UDP data. The default is 41,600 bytes. This attribute must specify a socket buffer size less than or equal to the setting of the sb_max attribute. This is a runtime attribute.

udp_sendspace

This specifies the system default socket buffer size for sending UDP data. The default is 9216 bytes. This attribute must specify a socket buffer size less than or equal to the setting of the sb_max attribute. This is a runtime attribute.

udp_ttl

This specifies the time to live for UDP packets. The default is 30 seconds. This is a runtime attribute.

udpcksum

This allows UDP checksum to be turned on/off. A value of 0 turns it off, while a value of 1 turns it on. Default is 1. udpcksum is a runtime attribute.

Note

If you use the tcp_recvspace, tcp_sendspace, udp_recvspace or udp_sendspace attribute to specify a socket to a buffer size larger than the sb_max attribute default, you must set the sb_max attribute to an equal or greater value. Otherwise, the socket system call returns the ENOBUFS error message when an application tries to create a socket.

C.2 Streams Tunable Attributes

The following Streams tunable attributes apply only for AIX Version 4.2 or later.

lowthresh

This specifies the maximum number of bytes that can be allocated using the allocb() call for the BPRI_LO priority. When the total amount of memory allocated by the net_malloc() call reaches this threshold, the allocb() request for the BPRI_LO priority returns 0. The lowthresh attribute represents a percentage of the thewall attribute, and you can set its value from 0 to 100.

This is a runtime attribute. Its default value is set to 90 (90 percent of thewall attribute).

medthresh

This specifies the maximum number of bytes that can be allocated using the allocb() call for the BPRI_MED priority. When the total amount of memory allocated by the net_malloc() call reaches this threshold, the allocb() request for the BPRI_MED priority returns 0. The medthresh attribute represents a percentage of the thewall attribute and you can set its value from 0 to 100.

This is a runtime attribute. Its default value is set to 95 (95 percent of thewall attribute).

nstrpush

This specifies the maximum number (should be at least 8) of modules that you can push onto a single Stream.

This is a loadtime attribute. Its default value is set to 8.

psebufcalls

This specifies the maximum number of bufcalls to allocate by Streams. The Stream subsystem allocates a certain number of bufcall structures at initialization, so that, when the allocb() call fails, the user can register their requests for the bufcall(). You are not allowed to lower this value until the system reboots, at which time it returns to its default value.

This is a runtime attribute. Its default value is set to 20.

pseintrstack

This specifies the maximum size of the interrupt stack allowed by Streams while running in the offlevel. Sometimes, when a process running other than the INTBASE level enters into a Stream, it encounters a stack overflow problem because the interrupt stack size is too small. Setting this attribute properly reduces the chances of stack overflow problems.

This is a loadtime attribute. Its default value is set to 0x3000.

psetimers

This specifies the maximum number of timers to allocate by Streams. The Stream subsystem allocates a certain number of timer structures at initialization so that the Streams driver or module can register its timeout() calls. You are not allowed to lower this value until the system reboots, at which time it returns to its default value.

This is a runtime attribute. Its default value is set to 20.

strctlsz

This specifies the maximum number of bytes of information that a single system call can pass to a Stream to place into the control part of a message (in an M_PROTO or M_PCPROTO block). A putmsg() call with a control part exceeding this size will fail with ERANGE.

This is a runtime attribute. Its default value is set to 1024.

strmsgsz

This specifies the maximum number of bytes of information that a single system call can pass to a Stream to place into the data part of a message (in M_DATA blocks). Any write() call exceeding this size is broken into multiple messages. A putmsg() call with a data part exceeding this size will fail with ERANGE.

This is a runtime attribute. Its default value is set to 1024.

strthresh

This specifies the maximum number of bytes Streams are normally allowed to allocate. When the threshold is passed, users without the appropriate privilege cannot open Streams, push modules, or write to Streams devices, and ENOSR is returned. The threshold applies only to the output side and does not affect data coming into the system (for example, the console continues to work properly). A value of 0 means that there is no threshold.

The strthresh attribute represents a percentage of the thewall attribute and you can set its value from 0 to 100. The thewall attribute indicates the maximum number of bytes that can be allocated by Streams and Sockets using the net_malloc() call. When you change thewall attribute, the threshold gets updated accordingly.

strturncnt

This specifies the maximum number of requests handled by the currently running thread for Module or Elsewhere level Streams synchronization.The Module level synchronization works in such a way that only one thread can run in the module at any time, and all other threads that try to acquire the same module will enqueue their requests and leave. After the currently running thread completes its work, it dequeues all the previously enqueued requests one by one and invokes them. If there are a large number of requests enqueued in the list, the currently running thread has to serve everyone and will always be busy serving others while starving itself. To avoid this, the currently running thread serves only the strturncnt number of threads, after which a separate kernel thread wakes up and invokes all the pending requests.

This is a runtime attribute. Its default value is set to 15.

C.3 Examples

1. To change the maximum size of the mbuf pool to 3 MB, enter:

   ```
   no -o thewall=3072
   ```

2. To reset the maximum size of the mbuf pool to its default size, enter:

   ```
   no -d thewall
   ```

3. To change the default socket buffer sizes on your system, add the following lines to the end of the /etc/rc.net file:

   ```
   /usr/sbin/no -o tcp_sendspace=16384
   ```

   and issue:

   ```
    /usr/sbin/no -o udp_recvspace=16384
   ```

4. To use an AIX machine as an Internet work router over TCP/IP networks, enter:

   ```
   no -o ipforwarding=1
   ```

C.4 Related Information

Network Overview for System Management in *AIX System Management Guide: Communications and Networks,* GC23-2487.

Monitoring and Tuning Communications I/O in *AIX Version 3.2 and 4.1 Performance Tuning Guide,* SC23-2365.

Appendix D. Special Notices

This publication is intended to help RS/6000 SP users, system administrators, and system engineers understand the available performance monitoring and system tuning tools on RS/6000 SP and to undertake a detailed performance analysis. The information in this publication is not intended as the specification of any programming interfaces that are provided by RS/6000 SP or POWERparallel System Support Programs. See the PUBLICATIONS section of the IBM Programming Announcement for RS/6000 SP and POWERparallel System Support Programs for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

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

ADSTAR	AIX
AS/400	AT
CT	IBM
LoadLeveler	Micro Channel
POWERparallel	PowerPC 604
PROFS	RS/6000
SP	SP2
System/390	3090
400	

Appendix E. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

E.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see "How to Get ITSO Redbooks" on page 285

- *Customizing Performance Toolbox and Performance Toolbox Parallel Extensions for AIX,* SG24-2011
- *Understanding IBM RS/6000 Performance and Sizing,* SG24-4810
- *RS/6000 Performance Tools in Focus,* SG24-4989
- *Benchmarking in Focus,* SG24-5052
- *AIX 64-bit Performance in Focus,* SG24-5103
- *Inside the RS/6000 SP,* SG24-5145
- *Understanding and Using the SP Switch,* SG24-5161

E.2 Redbooks on CD-ROMs

Redbooks are also available on the following CD-ROMs. **Order a subscription** and receive updates 2-4 times a year.

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 Redbook	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037

E.3 Other Publications

These publications are also relevant as further information sources:

- *RS/6000 SP Planning Volume 1, Hardware and Physical Environment*, GA22-7280
- *RS/6000 Planning Volume 2, Control Workstation and Software Environment*, GA22-7281
- *AIX Version 3.2 System Management Guide: Communications and Networks*, GC23-2487
- *PSSP: Performance Monitoring Guide and Reference*, SA22-7353
- *AIX Version 3.2 and 4.1 Performance Monitoring and Tuning Guide*, SC23-2365
- *AIX Performance Toolbox/6000 User's Guide Version 1.2 and 2.1*, SC23-2625
- *SP: Managing Shared Disks*, SC23-3849
- *AIX Performance Tuning Guide*, SR28-5930
- *IBM System Journal, Vol 34, No. 2,* 1995

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

ACL	Access Control List
AIX	Advanced Interactive Executive
API	Application Programming Interface
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
BIS	Boot/Install Server
BSD	Berkeley Software Distribution
CPU	Central Processing Unit
CSS	Communication Subsystem
CW	Control Workstation
DB	Database
FDDI	Fiber Distributed Data Interface
FIFO	First-In First-Out
GB	Gigabytes
GL	Group Leader
GPFS	General Purpose File System
GVG	Global Volume Group
HiPS	High Performance Switch
HSD	Hashed Shared Disk
IBM	International Business Machines Corporation
ICMP	Internet Control Message Protocol
IP	Internet Protocol
ISO	International Standards Organization
ITSO	International Technical Support Organization
JFS	Journaled File System
LAN	Local Area Network
LED	Light Emitting Diode
LVM	Logical Volume Manager
MB	Megabytes
MIB	Management Information Base
MPI	Message Passing Interface
MPL	Message Passing Library
MPP	Massive Parallel Processors
MTU	Maximum Transmission Unit
NFS	Network File System
NIM	Network Installation Manager
NSB	Node Switch Board
NSC	Node Switch Chip
OID	Object ID
ODM	Object Data Manager
PID	Process ID
PROFS	Professional Office System
PSSP	Parallel System Support Programs
PTC	Prepare To Commit
PTPE	Performance Toolbox Parallel Extensions
PTX/6000	Performance Toolbox/6000

RAM	Random Access Memory
RCP	Remote Copy Protocol
RPQ	Request for Product Quotation
RVSD	Recoverable Virtual Shared Disk
SDR	System Data Repository
SNMP	Simple Network Management Protocol
SP	RS/6000 SP

Index

ITSO Redbook Evaluation

RS/6000 SP System Performance Tuning
SG24-5340-00

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!)

SG24-5340-00
Printed in the U.S.A.

RS/6000 SP System Performance Tuning

SG24-5340-00