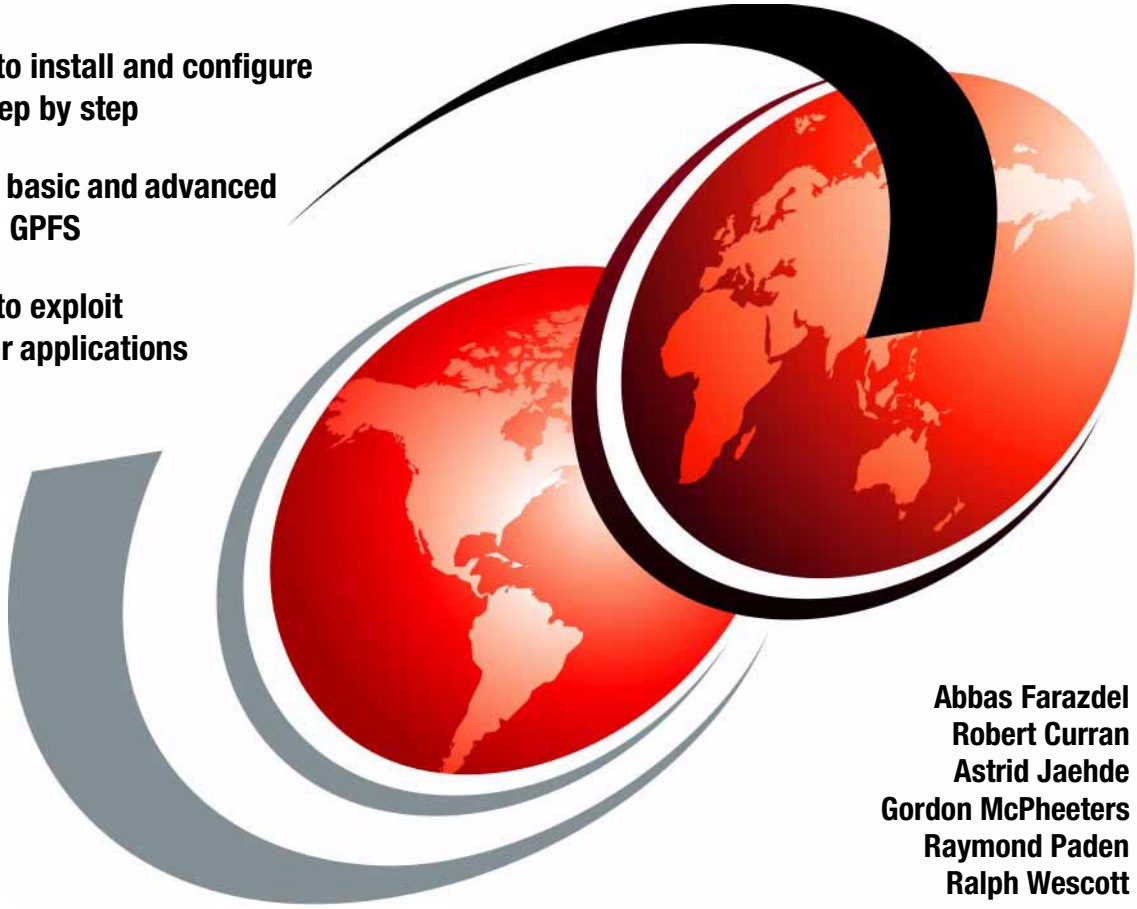
IBM

GPFS on AIX Clusters:

High Performance File System Administration Simplified

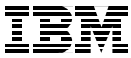
- Learn how to install and configure GPFS 1.4 step by step
- Understand basic and advanced concepts in GPFS
- Learn how to exploit GPFS in your applications



Abbas Farazdel
Robert Curran
Astrid Jaehde
Gordon McPheeters
Raymond Paden
Ralph Wescott

Redbooks

ibm.com/redbooks



International Technical Support Organization

GPFS on AIX Clusters: High Performance File System Administration Simplified

August 2001

SG24-6035-00

Take Note! Before using this information and the product it supports, be sure to read the general information in “Special notices” on page 265.

First Edition (August 2001)

This edition applies to Version 1 Release 4 of IBM General Parallel File System for AIX (GPFS 1.4, product number 5765-B95) or later for use with AIX 4.3.3.28 or later.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

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

Contents

Preface

With the newest release of General Parallel File System for AIX (GPFS), release 1.4, the range of supported hardware platforms has been extended to include AIX RS/6000 workstations that are not part of an RS/6000 SP system. This is the first time that GPFS has been offered to non-RS/6000 SP users. Running GPFS outside of the RS/6000 SP does require the high availablity cluster multi-processing/enhanced scalabilty (HACMP/ES) to be configured and the RS/6000 systems within the HACMP cluster (that will be part of the GPFS cluster) be concurrently connected to a serial storage architecture (SSA) disk subsystem.

This redbook focuses on the planning, installation and implementation of GPFS in a cluster environment. The tasks to be covered include the installation and configuration of HACMP to support the GPFS cluster, implementation of the GPFS software, and developing application programs that use GPFS. A troubleshooting chapter is added in case any problems arise.

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.

Abbas Farazdel is an SP/Cluster System Strategist, Technical Consultant, and Senior Project Manager at the International Technical Support Organization, Poughkeepsie Center. Before joining the ITSO in 1998, Dr. Farazdel worked in the Global Business Intelligence Solutions (GBIS) group at IBM Dallas as an Implementation Manager for Data Warehousing and Data Mining Solutions and in the Scientific and Technical Systems and Solutions (STSS) group at the IBM Thomas J. Watson Research Center as a High Performance Computing Specialist. Dr. Farazdel holds a Ph.D. in Computational Quantum Chemistry and an M.Sc. in Computational Physics from the University of Massachusetts.

Robert Curran is a Senior Technical Staff Member at the IBM Poughkeepsie UNIX Development Laboratory. He has worked in IBM software development for over 25 years. During most of this time, he has been involved in the development of database, file system, and storage management products. He holds an M. Sc. degree in Chemistry from Brown University. He is currently the development project leader for GPFS.

Astrid Jaehde is a software engineer at Availant, Inc., Cambridge, MA. She is a member of the HACMP development team. Astrid has five years of UNIX experience and holds a degree in Mathematics from Dartmouth College.

Gordon McPheeters is an Advisory Software Engineer with the GPFS Functional Verification and Test team based in IBM Poughkeepsie, NY. His background includes working as an AIX Systems Engineer for IBM Canada and for the IBM agent in Saudi Arabia. Prior to getting involved in AIX in 1991, he worked in the oil and gas industry in Western Canada as an MVS Systems Programmer.

Raymond Paden works for IBM as an I/O architect and Project Manager assisting customers with their I/O needs. Prior to joining IBM, he worked for six years as a team manager and systems programmer developing seismic processing applications and 13 years as a professor of Computer Science. He holds a Ph.D. in Computer Science from the Illinois Institute of Technology. His areas of technical expertise include disk and tape I/O, performance optimization and operating systems on parallel systems such as the IBM SP system. He has written on topics including parallel and combinatorial optimization, and I/O.

Ralph Wescott is an SP Systems Administrator working for Pacific Northwest National Laboratory in Washington State. He holds a BS degree from the State University of New York. During his almost 20 year career in IBM Ralph was a Customer Engineer, Manufacturing Engineer and a Systems Engineer. His areas of expertise include UNIX, RS/6000, SP, GPFS, and anything hardware.

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

International Technical Support Organization, Austin Center

Matthew Parente

IBM Poughkeepsie

Myung Bae
Kuei-Yu Wang-Knop

Availant Inc., Cambridge

Jim Dieffenbach
John Olson
Venkatesh Vaidyanathan

Special notice

This publication is intended to help system administrators, analysts, installers, planners, and programmers of GPFS who would like to install and configure GPFS 1.4. The information in this publication is not intended as the specification of any programming interfaces that are provided by GPFS or HACMP/ES. See the PUBLICATIONS section of the IBM Programming Announcement for GPFS Version 1, Release 4 and HACMP/ES Version 4, Release 4 for more information about what publications are considered to be product documentation.

IBM trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX
AS/400
Current
IBM ®
Notes
Redbooks Logo 
SP
XT
AT
CT
e (logo)® 
Micro Channel
Redbooks
RS/6000
SAA
SP2

Comments welcome

Your comments are important to us!

We want our IBM Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to the address on page ii.

1

A GPFS Primer

This introductory chapter briefly describes topics which should be understood prior to attempting the first installation of the GPFS product. It includes the following:

- What is GPFS
- Why GPFS
- The basics
- When to consider GPFS
- Planning considerations
- The application view

1.1 What is GPFS

The General Parallel File System (GPFS) for AIX provides global access to data from any of the hosts within a cluster or within an RS/6000 SP. It is IBM's first shared disk file system. It was initially released on the RS/6000 SP in 1998 using a software simulation of a storage area network called the IBM Virtual Shared Disk (VSD). The VSD provides the transport of disk data blocks across either IP or a high performance protocol private to the SP switch. This is similar in function to many initiatives within the computer industry to access disk blocks across IP networks. The product has been running on cluster configurations of up to 512 nodes supporting demanding I/O loads and fault tolerance requirements since its introduction.

GPFS file systems support multiple tera bytes of storage within a single file system. As the hardware technology supporting storage attachment matures, it is a logical extension of the GPFS environment to support disks that have a shared direct attachment. GPFS 1.4 begins that process by introducing support for clusters of IBM @server pSeries and IBM RS/6000 machines running HACMP and sharing access to disks through SSA links.

At its core, GPFS is a parallel disk file system. The parallel nature of GPFS guarantees that the entire file system is available to all nodes within a defined scope and the file system's services can be safely applied to the same file system on multiple nodes simultaneously. It also means that multiple records can be safely written to the same file simultaneously by being striped across several disks (thus improving performance). This GPFS parallel feature can improve the performance of both parallel and sequential programs. In other words, you do not have to write parallel I/O code to benefit from GPFS's parallelization. GPFS will automatically parallelize the I/O in your sequential program.

In addition to its parallel features, GPFS supports high availability and fault tolerance. The high availability nature of GPFS means that the file system will remain accessible to nodes even when a node in the file system dies. The fault tolerant nature of GPFS means that file data will not be lost even if some disk in the file system fails. These features are provided through integration with other software and hardware, for example, HACMP/ES, and RAID.

1.2 Why GPFS

GPFS enables users to group applications based on a business need rather than the location of the data.

Specifically, GPFS allows:

- The consolidation of applications that share data on a cluster of pSeries or RS/6000 server nodes or RS/6000 SP nodes. You no longer need to move applications to data.
- The creation of multi-machine clusters for situations where the applications require more computing resources than are available in one server or require that processing be available from a backup server in the event of the failure of the primary server. GPFS provides high performance data access from multiple servers with proven scalability and can be configured to allow continuous access to data with the failure of one or more nodes or their disk attachment capabilities.
- The execution of parallel applications that require concurrent sharing of the same data from many nodes in the complex, including concurrent updates of all files. In addition, GPFS provides extended interfaces that can be used to provide optimal performance for applications with difficult data access patterns.
- Parallel maintenance of metadata, thus offering higher scalability and availability.

GPFS provides high speed access data from any of the nodes of the HACMP cluster or the SP through normal application interfaces; no special programming is required. It performs all file system functions including metadata functions on all members of the cluster. This is in contrast to other storage area network (SAN) file systems, which have centralized metadata processing nodes for each file system and can become a performance bottleneck. This allows the designer of the system containing GPFS to allocate applications to compute resources within the cluster without concern for which member of the cluster has the data. GPFS provides recovery capabilities so that the failure of a machine will not cause the loss of data access for the remaining machines.

1.3 The basics

This redbook is written for GPFS Version 1, Release 4; however, much of the information applies to prior releases of GPFS operating on the RS/6000 SP. GPFS 1.4 provides concurrent high speed file access to applications executing on multiple nodes of an RS/6000 SP system or on multiple systems that form an HACMP cluster. The support for HACMP clusters which are not executing on an SP is new with release 4. This chapter will address topics that should be understood prior to attempting the first installation of the product. It assumes that the reader has a basic knowledge of either the RS/6000 SP with its associated software or the HACMP environment.

We will use the term *cluster* to describe either the nodes of an SP or the members of an HACMP cluster that shares an instance of GPFS. We will use the term *direct attach* to describe disks that are physically attached to multiple nodes using SSA connections and contrast that to the VSD connections within the SP. Figure 1-1 shows a cluster residing on an SP using the VSD. Figure 1-2 on page 7 shows a similar cluster using directly attached disks.

GPFS is targeted at applications which execute on a set of cooperating cluster nodes running the AIX operating system and shares access to the set of disks that make up the file system. These disks may be physically shared using SSA loops directly attached to each node within HACMP clusters or shared through the software simulation of a storage area network provided by the IBM Virtual Shared Disk and the SP switch. Consult the latest IBM product documentation for additional forms of physically shared connectivity.

In addition, GPFS requires a communication interface for the transfer of control information. This interface does not need to be dedicated to GPFS; however, it needs to provide sufficient bandwidth to meet your GPFS performance expectations. On the SP, this interface is the SP switch (SP Switch or SP Switch2). For HACMP clusters, we recommend a LAN with a capability of at least 100Mb/sec.

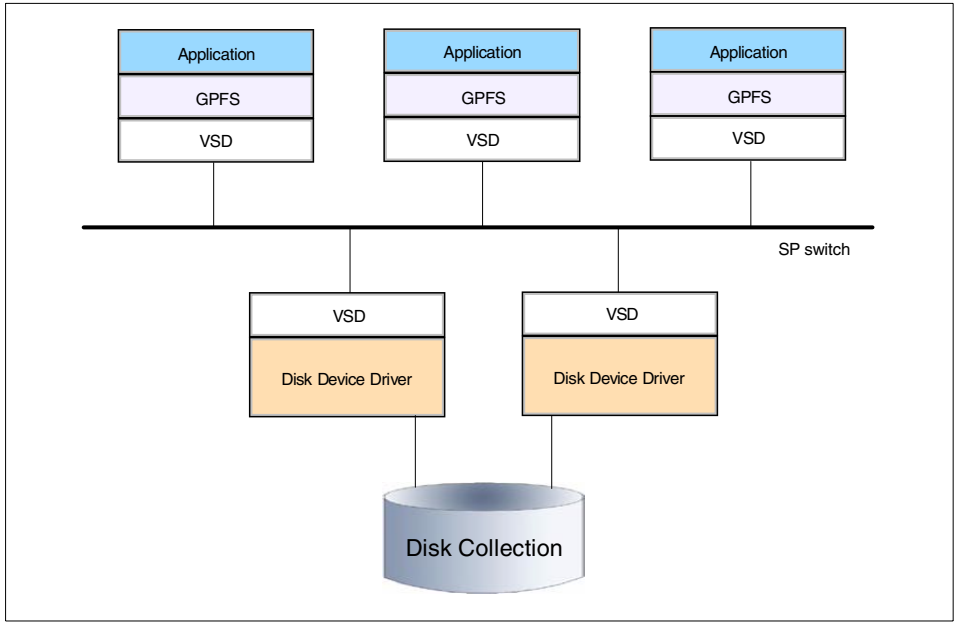


Figure 1-1 A simple GPFS configuration with VSD on an SP

GPFS provides excellent performance for each of the following classes of applications:

- Parallel applications that require shared access to the same file from multiple application instances,
- Batch serial applications that are scheduled to available computing resources and require access to data on the available machine, or
- Simple applications that in their aggregate require the performance/reliability of a cluster of machines.

GPFS is designed to provide a common file system for data shared among the nodes of the cluster. This goal can be achieved using distributed file systems such as NFS, but this often provides less performance and reliability than GPFS users require. GPFS provides the universal access that applications need with excellent performance and reliability characteristics. The basic characteristics of GPFS are:

- It is an AIX style file system, that means most of your applications work without any change or recompile.
- It provides access to all GPFS data from all nodes of the cluster. GPFS will provide best performance for larger data objects, but can also provide benefits for large aggregates of smaller objects.
- It can be configured with multiple copies of metadata allowing continued operation should the paths to a disk or the disk itself be broken. Metadata is the file system data that describes the user data. GPFS allows the use of RAID or other hardware redundancy capabilities to enhance reliability.
- The loss of connectivity from one node to the storage does not affect the others in the direct storage attachment configurations. In SP configurations which use VSD and the SP switch, the capability is provided to route disk data through multiple VSD servers allowing redundancy. In either configuration, the loss of one node does not cause a total loss of access to file system data.
- In the direct attach configurations, the data performance is that of the SSA connections between storage and the systems. Multiple SSA links can be configured up to the maximum number of the adapter slots available in the node selected. All of these SSA links can be applied to a single file system, if necessary.

  On the SP, that IBM Virtual Shared Disk (VSD) facility and the SP switch fabric provides low overhead data movement between a node that has a physical connection to a disk and an application node requiring the data on the disk. The disks can be spread across a large number of adapters within a server and across a large number of servers in order to generate very high performance while accessing a single file system or a number of file systems.

- It uses the Group Services component of the IBM Parallel Systems Support Program (PSSP) or HACMP to detect failures and continue operation whenever possible.
- Release 4 of GPFS uses SSA direct multi-attachment of disks. Additional direct attachment methods are possible in the future. On the SP, VSD allows the use of any type of disk which attaches to the RS/6000 SP and is supported by AIX.
- GPFS data can be exported using NFS including the capability to export the same data from multiple nodes. This provides potentially higher throughput than servers that are limited to one node. GPFS data can also be exported using DFS although the DFS consistency protocols limit the export to one node per file system.

Figure 1-1 on page 4 illustrates a simple five node GPFS configuration. The three nodes at the top of the configuration are home to applications using GPFS data. The two at the bottom share connections to some number of disks. One of these VSD servers is the primary path for all operations involving each disk, but the alternate path is used if the primary is not available. A node can be the primary for some disks and the backup for others.

GPFS uses a token manager to pass control of various disk objects among the cooperating instances. This maintains consistency of the data and allows the actual I/O path to be low function and high performance. Although we have illustrated applications and VSD servers on independent nodes, they can also share a node. The VSD servers consume only a portion of the CPU cycles available on these nodes, and it is possible to run some applications there. The GPFS product documentation describes these choices in more detail.

The use of the backup disk server covers the failure of a single VSD server node. The failure of individual disk drives can cause data outages. However, the use of RAID, AIX mirroring, or GPFS replication can mitigate these outages. GPFS also provides extensive recovery capabilities that maintain metadata consistency across the failure of application nodes holding locks or performing services for other nodes. Reliability and recovery have been major objectives of the GPFS product from its inception.

Figure 1-2 on page 7 shows a GPFS configuration within an HACMP cluster. The HACMP cluster differs from the SP cluster in that it requires direct attachment of the SSA disks to every node. It also requires the communications link shown to carry control information such as tokens between nodes. The SSA adapters support two nodes in RAID mode or eight nodes in JBOD (“just a bunch of disks”) mode, so GPFS replication of data may be useful in larger configurations.

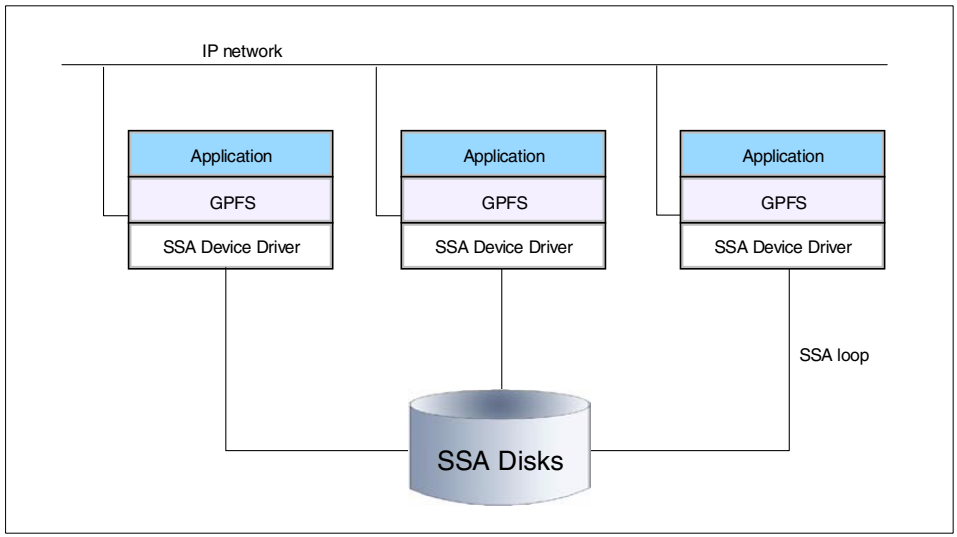


Figure 1-2 GPFS in an HACMP environment

1.4 When to consider GPFS

There are several situations where GPFS is the ideal file system for data on the SP or a cluster of RS/6000 machines.

- You have large amounts of file data which must be accessed from any node and where you wish to more efficiently use your computing resources for either parallel or serial applications.
- The data rates required for file transfer exceed what can be delivered with other file systems.
- You require continued access to the data across a number of types of failures.

GPFS is not a wide area distributed file system replacing NFS or DFS for network data sharing, although GPFS files can be exported using NFS or DFS.

1.5 Planning considerations

This section will identify a few areas to think about before installing GPFS. The detailed information behind this section is in the *GPFS for AIX: Concepts, Planning, and Installation Guide*.

There are four steps to be taken before attempting a GPFS installation. We will overview the thought process and considerations in each of these steps:

- Consider your I/O requirements
- Plan your hardware layout
- Consider the GPFS prerequisites
- Consider the GPFS parameters required to meet your needs

1.5.1 I/O requirements

All of the steps which follow presume some knowledge of your applications and their demands on I/O. This step is almost always imperfect, but better knowledge leads to better results.

The following questions may help in thinking about the requirements:

- What are the parallel data requirements for your applications? If they are parallel, are they long running? Running applications with higher parallelism and shared data may generate requirements for LAN/switch bandwidth.
- What requirements do you have for the number of applications running concurrently? What I/O rate do they generate? I/O rates should be specified as either bytes/sec or number of I/O calls/sec whichever is the dominant need. You must provide sufficient disk devices to sustain the required number of I/Os or bytes/sec.
- How many files do you have and what size are they? There are GPFS parameters which will optimize towards large or smaller files.
- What types of access patterns do your applications use? Are they random or sequential or strided? Do they re-access data such that increased caching might be useful?

With some thoughts on these topics, you will be better prepared to specify a successful GPFS system and the hardware required.

1.5.2 Hardware planning

GPFS uses a number of hardware resources to deliver its services. The proper configuration of these resources will result in better performance and reliability.

You should consider the number of disks required to supply data to your application. Disks need to be measured both in terms of capacity and in terms of I/O speed. When considering the I/O speed of your devices, you should be looking at the specification of random I/O at the block size of the file system or

the request size of your dominant applications, whichever is larger. This is not the same as the burst I/O rate quoted by disk manufacturers. As file systems fragment, disk blocks related to the same file will get placed where space is available on the disks.

You should consider if RAID is needed for your systems and if so, match the RAID stripe width to the block size of the file system. GPFS and other file systems perform I/O in file system block multiples and the RAID system should be configured to match that block size unless the application set is mostly read only.

You should consider the disk attachment mechanism and its capabilities. In general, both SSA and SCSI allow the attachment of more disks than the links can transfer at peak rates in a short period. If you are attaching disks with an objective for maximum throughput from the disks, you will want to limit the number of disks attached through any adapter. If disk capacity, rather than optimal transfer rates, is the major concern, more disks can use the same adapter. If you are operating in direct attach mode, note that the disk attachment media is shared among all the nodes and you should plan on enough disk attachment media to achieve the desired performance.

If you are configuring your GPFS with VSDs you should consider the number of VSD servers required to achieve your expected performance. The VSD server performance is usually limited by the capabilities of the specific server model used. The limiting factor is usually some combination of the I/O bandwidth of the node and the LAN/switch bandwidth available to the node. CPU utilization of VSD servers is usually relatively low. The capabilities of differing node types will vary. Spreading the load across additional VSD servers may also be beneficial if I/O demand is very bursty and will cause temporary overloads at the VSD servers.

In a VSD environment, the decision to run applications on the VSD servers or to have dedicated VSD servers is primarily dependent on the nature of your applications and the amount of memory on the nodes. There are additional CPU cycles on most VSD servers that can be used by applications. The requirements of VSD service are high priority to be responsive to I/O devices. If the applications are not highly time sensitive or highly coupled with instances that run on nodes which do not house VSD servers, use of these extra cycles for applications is feasible. You should insure that sufficient memory is installed on these nodes to meet the needs of both the disk/network buffers required for VSD and the working area of the application.

1.5.3 GPFS prerequisites

A detailed discussion of pre-requisites is beyond the scope of this chapter; but the following checklists might be useful in anticipating the requirements for GPFS.

On the SP

- GPFS administration uses the PSSP security facilities for administration of all nodes. You should insure that these are correctly set up. The *PSSP: Administration Guide,* SA22-7348 describes this.
- GPFS uses the VSD and requires that it be configured correctly. Be sure to consider the number of pbufs and the number and size of buddy buffers that you need. The *PSSP: Managing Shared Disks,* SA22-7349 publication describes these topics.
- GPFS uses the Group Services and Topology Services components of the PSSP. In smaller systems, the default tuning should be acceptable for GPFS. In larger configurations, you may wish to consider the correct values of frequency and sensitivity settings. See the *PSSP: Administration Guide,* SA22-7348 for this information.

In HACMP clusters

- GPFS uses an IP network which connects all of the nodes. This is typically a LAN with sufficient bandwidth available for GPFS control traffic. A minimum bandwidth of 100 Mb/sec is required.
- GPFS requires that the SSA disks be configured to all of the nodes within the GPFS cluster.
- GPFS uses the Group Services and Topology Services components of HACMP/ES.
- You may not use LVM mirroring or LVM bad block relocation.

1.5.4 GPFS parameters

The major decisions in the planning of GPFS involve the file system block sizes and the amount of memory dedicated to GPFS. Using larger block sizes will be beneficial for larger files because it will more efficiently use the disk bandwidth. Use of smaller blocks sizes will increase the effectiveness of disk space utilization for a workload that is dominated by large numbers of small files. It may also increase cache memory utilization if the workload contains few files of a size above one file system block. See the product documentation for more information.

GPFS, like all file systems, caches data in memory. The cache size is controlled by a user command and is split into two pieces; space for control information and space for file data. Increasing the amount of space available may increase the performance of many workloads. Increasing it excessively will cause memory shortages for other system components. You may wish to vary these parameters and observe the effects on the overall system.

1.6 The application view

GPFS provides most of the standard file system interfaces so many applications work unchanged. The only caution applies to parallel applications which update the same file concurrently.

Some consideration of the partitioning of data in parallel applications will be valuable because GPFS on each node must deal with file system blocks and disk sectors. Optimal performance will be obtained from parallel applications, doing updates of the same file, if they respect these system capabilities. If each task of the parallel application were to operate on a series of consecutive file system blocks, better throughput will be obtained than if multiple tasks updated data within the same file system block or disk sector. Parallel applications coded in this fashion also take advantage of the file systems ability to overlap needed data fetches with application execution because sequential access allows prediction of what data is needed next.

2

More about GPFS

This chapter provides a high level, conceptual framework for GPFS by describing its architecture and organization. It is particularly applicable to system administrators who install, configure and maintain GPFS and to systems programmers who develop software using GPFS. Chapter 8, “Developing Application Programs that use GPFS” on page 143 explores GPFS in greater depth from an application programming perspective. While GPFS can run in a number of different environments, this chapter pursues its discussion assuming that GPFS is running in a clustered environment with directly attached disks. Chapter 3, “The cluster environment” on page 27 discusses clustering explicitly in greater detail. Many of the concepts discussed in this chapter are discussed in greater depth with broader scope in *GPFS for AIX: Concepts, Planning, and Installation Guide*.

In particular, this chapter discusses:

- An overview of GPFS’s structure and environment
- GPFS’s global management functions
- GPFS file architecture
- GPFS’s use of memory, with an emphasis on caching

2.1 Structure and environment

GPFS is designed to work in an AIX environment. AIX provides basic operating system services and a programmer visible API (e.g., open(), read(), write()) which acts on behalf of the application program to access the GPFS data processing calls. In a clustered environment, AIX also provides the logical volume manager (LVM) for concurrent disk management and configuration services. In addition to these AIX services, GPFS needs services provided by HACMP/ES and Group Services in a clustered environment. HACMP/ES provides the basic cluster functionality that supports GPFS's mode of concurrent I/O operation. Group Services provides process failure notification, and recovery sequencing on multiple nodes. Figure 2-1 illustrates this architecture.

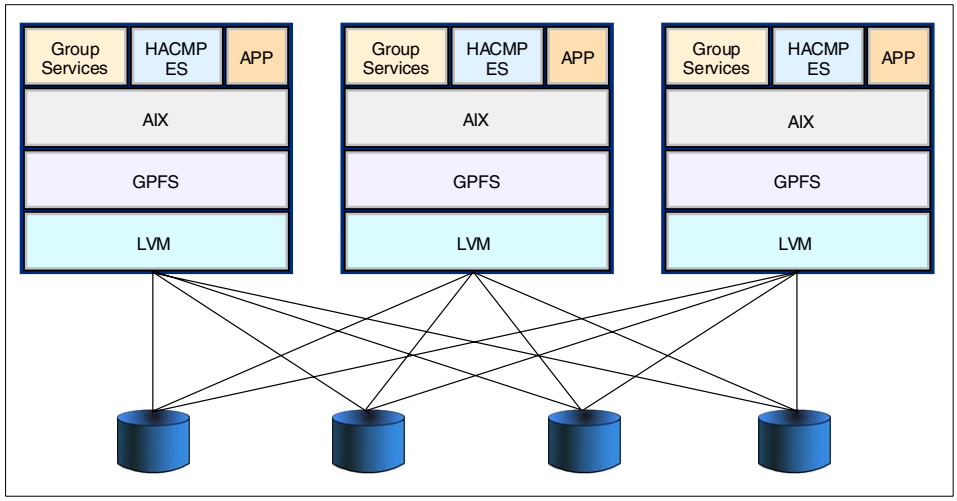


Figure 2-1 GPFS architecture

In this setting, HACMP/ES defines a cluster of nodes. A GPFS cluster then resides within the HACMP/ES cluster. The GPFS cluster defines a single distributed scope of control for collectively maintaining a set GPFS file systems. There can be one or more *nodesets* within the GPFS cluster. A nodeset is a set of nodes over which a particular file system is visible. Figure 2-2 illustrates the relationship between these entities. Chapter 3, "The cluster environment" on page 27 discusses clustering in greater detail.

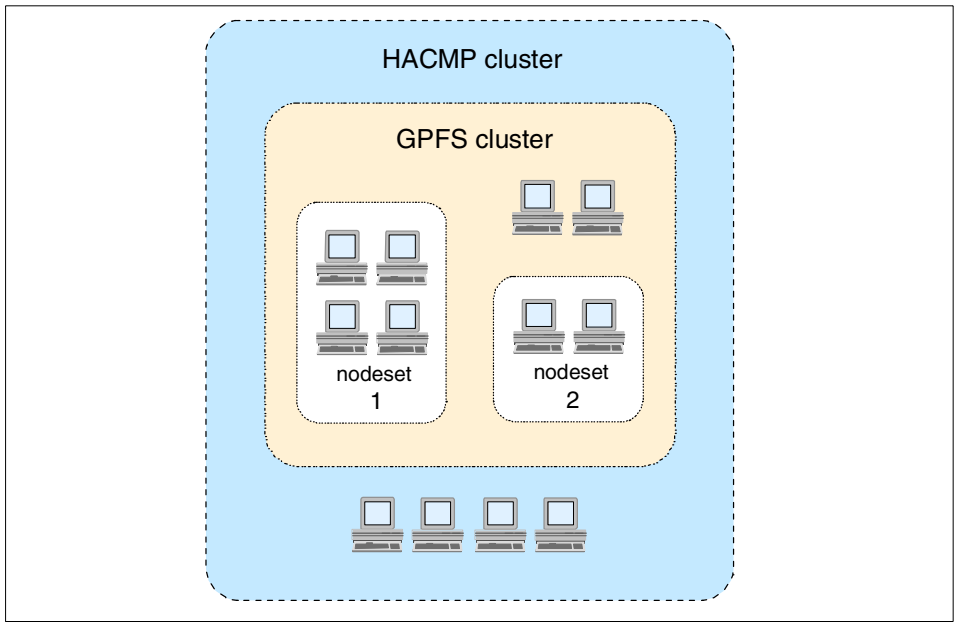


Figure 2-2 HACMP cluster, GPFS cluster and nodeset relationship

Structurally, GPFS resides on each node as a multi-threaded daemon (called *mmfsd*) and a kernel extension. The GPFS daemon performs all of the I/O and buffer management. This includes such things as *read-ahead* for sequential writes, *write-behind* for writes not declared to be synchronous and *token management* to provide atomicity and data consistency across multiple nodes. Separate threads from each daemon are responsible for some of these and other functions. This prevents higher priority tasks from being blocked. Application programs use the services of the GPFS kernel extension by making file system calls to AIX, which in turn presents the request to GPFS. Thus GPFS appears as another file system. In addition to the GPFS daemon and kernel extension, system administration commands are also available on all nodes.

From another perspective, GPFS can be viewed as a *client* allocating disk space, enforcing quotas, subdividing file records into blocks, guaranteeing I/O operations are atomic, and so forth. It is the combined actions of HACMP/ES, LVM and the disk hardware (e.g., SSA) that act as the *server* providing connectivity between the nodes and to the disk.

With this structure and environment, it is evident that GPFS consists of numerous components and that it interacts with components from many other systems; it is not a monolithic entity. Yet GPFS must coordinate its activities between all of these components. To do so, it communicates using *sockets*. In particular, user commands communicate with GPFS daemons using sockets and daemons communicate among themselves using sockets.

2.2 Global management functions

For most services GPFS performs the same types of activities on all nodes. For instance, all GPFS nodes in the cluster execute GPFS kernel extension system calls and schedule GPFS daemon threads to transfer data between the GPFS cache and disk. However, there are three management functions performed by GPFS globally from one node on behalf of the others, which are implemented by software components in GPFS. Because these functions are associated with a particular node, the nodes assume the function names. They are called:

- The configuration manager node
- The file system manager node
- The metanode

2.2.1 The configuration manager node

The configuration manager node selects the file system manager node and determines whether a *quorum* of nodes exist. A quorum in GPFS is the minimum number of nodes needed in a nodeset for the GPFS daemon to start. For nodesets with three or more nodes, the quorum size must be one plus half the nodes in the nodeset (called a *multi-node* quorum). For a two node nodeset one can either have a multi-node quorum or a *single-node* quorum. There is one configuration manger per nodeset.

In a multi-node quorum, if GPFS fails on a node (i.e., the GPFS daemon dies), it tries to recover. However, if the quorum is lost, GPFS recovery procedures restart its daemons on all GPFS nodes and attempt to re-establish a quorum. If the nodeset has only two nodes, then losing one of the nodes will result in the loss of quorum and GPFS will attempt to restart its daemons on both nodes. Thus three nodes in a nodeset is necessary to prevent shutting down the daemons on all nodes prior to re-starting them.

Alternatively, one can specify a single-node quorum when there are only two nodes in the nodeset. In this case, a node failure will result in GPFS *fencing* the failed node and the remaining node will continue operation. This is an important consideration since a GPFS cluster using RAID can have a maximum of two nodes in the nodeset. This two-node limit using RAID is an SSA hardware limitation.

2.2.2 The file system manager node

The file system manager node performs a number of services including:

- File system configuration
- Disk space allocation management
- Token management
- Quota management
- Security services

There is only one file system manager node per file system and it services all of the nodes using this file system. It is the configuration manager node's role to select the file system manager node. Should the file system manager node fail, then the configuration manager node will start a new file system manager node and all functions will continue without disruption.

It should be noted that the file system manager node uses some additional CPU and memory resources. Thus it is sometimes useful to restrict resource intensive applications from running on the same node as the file system manger node. By default, all nodes in a nodeset are eligible to act as the file system manager node. However, this can be changed by using **`mmchconfig`** to declare a node as ineligible to act as the file system manager node (see the *GPFS for AIX: Problem Determination Guide,* GA22-7434).

2.2.3 Metanode

For each open file, one node is made responsible for guaranteeing the integrity of the metadata by being the only node that can update a file's metadata. This node is called the metanode. The selection of a file's metanode is made independently of other files and is generally the node that has had the file open for the longest continuous period of time. Depending on an applications execution profile, a file's metanode can migrate to other nodes.

2.3 File structure

The GPFS file system is simply a UNIX file system with its familiar architecture, but adapted for the parallel features of GPFS. Thus a GPFS file consists of *user data* and *metadata* with *i-nodes* and *indirect blocks, striped* across multiple disks.

2.3.1 Striping

Striping is one of the unique features of GPFS compared with many native UNIX file systems such as the Journaled File System (JFS) under AIX. The purpose of striping is to improve I/O operation performance by allowing records to be automatically subdivided and simultaneously written to multiple disks; we will sometimes refer to this as *implicit parallelism* as the application programmer does not need to write any parallel code; all that is needed is to access records that are larger than a block.

The fundamental granularity of a GPFS I/O operation is generally, but not always, the *block,* sometimes called a *stripe*. The size of this block is set by the **`mmcrfs`** command. The choices are 16K, 64K, 256K, 512K, or 1024K (K represents 1024 bytes, or one kilobyte) and it cannot be *arbitrarily* changed once set (see **`man`** pages for **`mmcrfs`** and **`mmchconfig`**). For example, suppose the block size is 256K and an application writes a 1024K record. Then this record is striped over four disks by dividing it into four 256K blocks and writing each block to a separate disk at the same time. A similar process is used to read a 1024K record.

The expression *"granularity of a GPFS I/O" operation* refers to the smallest unit of transfer between an application program and a disk in GPFS file system. Generally this is a block. Moreover, on disk, blocks represent the largest contiguous chunk of data. However, because files may not naturally end on a block boundary, a block can be divided into 32 subblocks. In some circumstances, a subblock may be transferred between disk and an application program making it the smallest unit of transfer. Section 8.3.1, "Blocks and striping" on page 148 explains this in greater detail.

The choice of a block size is largely dependent upon a system's job profile. Generally speaking, the larger the block, the more efficient the I/O operations are. But if the record size in a typical transaction is small, while the block is large, much of the block is not being utilized effectively and performance is degraded. Perhaps the most difficult job profile to match is when the record size has a large variance. In the end, careful benchmarking using realistic workloads or synthetic benchmarks (see Appendix G, "Benchmark and Example Code") that faithfully simulate actual workloads are needed to properly determine the optimal value of this parameter.

Blocks can be striped in three ways. The default and most common way is *round robin* striping. In this method, blocks are written to the disks starting with a randomly selected disk (called the *first disk* in this chapter) and writing successive blocks to successive disks; when the last disk has been written to, the process repeats beginning with the first disk again. For example (refer to Figure 2-3), suppose you have 16 disks (disk0 to disk15) and the first disk chosen is disk13; moreover, you are writing the first record, it is 1024K and it starts at seek offset 0. It is then divided into 4 blocks (b_0, b_1, b_2, b_3) and is written to disk13, disk14, disk15, and disk0. Suppose that the second record written is 1024K and is written beginning at seek offset 3145728 (i.e., 3072K). It to, is divided into 4 blocks, but is written to disk1, disk2, disk3, and disk4.

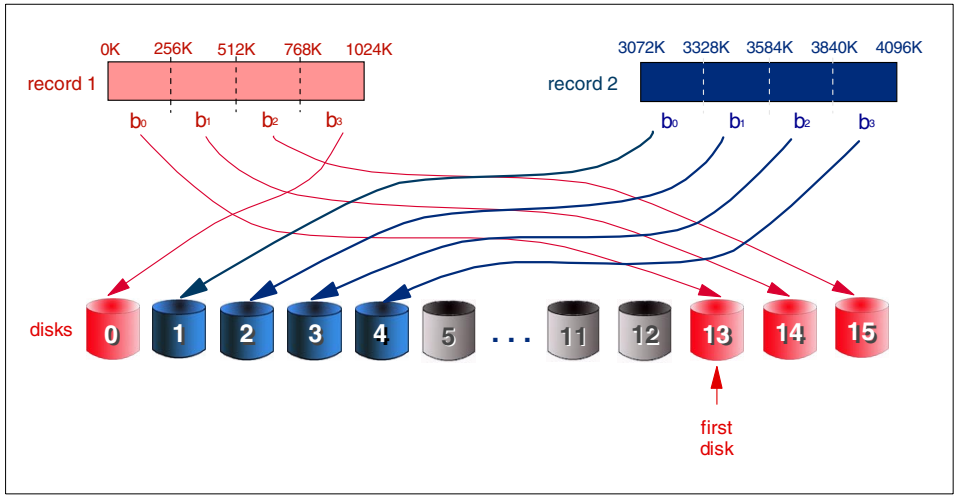


Figure 2-3 Round robin striping

The other methods are *random* and *balanced random*. Using the random method, the mapping of blocks to disks is simply random. With either the round robin or random method, disks are assumed to be the same size (if disks are not the same size, space on larger volumes is not wasted, but it is used at a lower throughput level). If the disks are not the same size, then the balanced random method randomly distributes blocks to disks in a manner proportional to their size. This option can be selected and changed using the **`mmcrfs`** and **`mmchfs`** commands.

Striping significantly impacts the way application programs are written and is discussed further in Chapter 8, “Developing Application Programs that use GPFS” on page 143.

2.3.2 Metadata

Metadata is used to locate and organize user data contained in GPFS's striped blocks. There are two kinds of metadata, i-nodes and indirect blocks.

An i-node is a file structure stored on disk. It contains *direct pointers* to user data blocks or pointers to indirect blocks. At first, while the file is relatively small, one i-node can contain sufficient direct pointers to reference the entire file's blocks of user data. But as the file grows, one i-node is insufficient and more are needed; these extra blocks are called indirect blocks. The pointers in the i-node become *indirect pointers* as they point to indirect blocks that point to other indirect blocks or user data blocks. The structures of i-nodes and indirect blocks for a file is represented as a tree with a maximum depth of four where the tree leaves are the user data blocks. Figure 2-4 illustrates this.

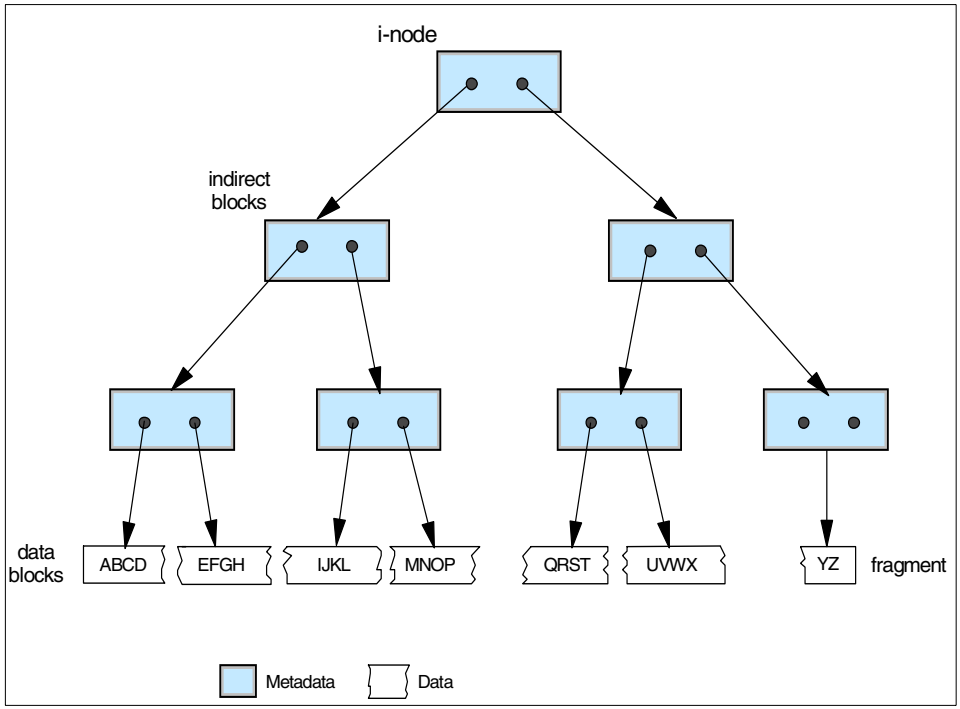


Figure 2-4 File system tree

Periodically when reading documentation on GPFS, you will encounter the term *vnode* in relation to metadata. A vnode is an AIX abstraction level above the i-node. It is used to provide a consistent interface for AIX I/O system calls, such as read() or write(), to the i-node structures of the underlying file system. For example, i-nodes for JFS and GPFS are implemented differently. When an application programmer calls read() on a GPFS file, a GPFS read is then initiated while the vnode interface gathers the GPFS i-node information.

2.3.3 User data

User data is the data that is read and used directly, or generated and written directly by the application program. This is the data that is striped across the disks and forms the leaves of the i-node tree. It constitutes the bulk of the data contained on disk.

2.3.4 Replication of files

While it is generally the case that most shops have only one copy of disk data on disk, GPFS provides a mechanism for keeping multiple copies of both user data and metadata. This is called *replication* and each copy is stored in separate *failure groups*. A failure group is a set of disks sharing a common set of adaptors; a failure with any component in the failure group can render the data it contains inaccessible. But storing each replica in a separate failure group guarantees that no one component failure will prevent access to the data (see also Chapter 3, “The cluster environment” on page 27). But it is not necessary to replicate both. You can, for example, replicate only the metadata so that in the event of a disk failure you can reconstruct the file and salvage the remaining user data. Such a strategy is used to reduce the cost of replication in shops with large volumes of data generated in short time periods.

2.3.5 File and file system size

GPFS is designed to support large files and large file systems. For instance, there is *no* two gigabyte file size limit in GPFS as is common on many other file systems. In terms of specific numbers, the current maximums for GPFS 1.4 are listed below.

- 32 file systems per nodeset
- 9 terabytes per file and per file system

  While the maximum limit is much larger, these are the largest file and file system sizes supported by IBM Service.
- 256 million files per file system

This is the architectural limit. The actual limit is set by using the **mmcrfs** command. Setting this value unrealistically high unnecessarily increases the amount of disk space overhead used for control structures.

If necessary, limits on the amount of disk space and number of files can be imposed upon individual users or groups of users through *quotas*. GPFS quotas can be set using the **mmedquota** command. The parameters can set soft limits, hard limits and grace periods.

Finally, a common task is to determine the size of a file. It is customary in a UNIX environment to use the **ls -l** command to ascertain the size of a file. But this only gives the *virtual* size of the file's user data. For example, if the file is sparsely populated, the file size reported by **ls -l** is equal to the seek offset of the last byte of the file. By contrast, the **du** command gives the size of the file in blocks, including its direct blocks. For sparse files, the difference in values can be significant. Example 2-1 illustrates this. sparse.file was created with a 1 megabyte record written at the end of it (i.e., at seek offset 5367660544). Doing the arithmetic to convert to common units, **ls -l** lists the file size as 5120 megabytes while **du -k** lists the file as just over 1 megabyte (i.e., 1.008; the extra .008 is for direct blocks). Example 2-2 illustrates the same commands on a dense file. Again, **ls -l** lists the file size as 5120 megabytes, but so does **du -k**. Now consider **df -k** in the two examples. In each case, /gpfs1 is the GPFS file system and contains only the one file listed by **ls**. Comparing **df -k** between the two examples shows that it accounts for the real file size as does **du -k**. (The same is also true for **mmdf**).

Example 2-1 Sparse file

```
host1t:/> ls -l /gpfs1/sparse.file
-rwxr-xr-x   1 root     system   5368709120 Feb 14 13:49 /gpfs1/sparse.file
host1t:/> du -k /gpfs1/sparse.file
1032    /gpfs1/sparse.file
host1t:/> df -k /gpfs1
Filesystem    1024-blocks      Free %Used    Iused %Iused Mounted on
/dev/gpfs1      142077952 141964544    1%       13     1% /gpfs1
```

Example 2-2 Dense file

```
host1t:/> ls -l /gpfs1/dense.file
-rwxr-xr-x   1 root     system   5368709120 Feb 14 13:49 /gpfs1/dense.file
host1t:/> du -k /gpfs1/dense.file
5242880 /gpfs1/dense.file
host1t:/> df -k /gpfs1
Filesystem    1024-blocks      Free %Used    Iused %Iused Mounted on
/dev/gpfs1      142077952 136722432    4%       13     1% /gpfs1
```

2.4 Memory utilization

GPFS uses memory in three forms:

- Kernel heap
- Daemon segments
- Shared segments

Memory from the kernel heap is allocated most generally for control structures that establish GPFS/AIX relations such as vnodes. The largest portion of daemon memory is used by file system manager functions to store structures needed for command and I/O execution. The shared segments are accessed both by the GPFS daemon and the kernel and form a *GPFS cache*. They are directly visible to the user and are more complex.

2.4.1 GPFS Cache

Shared segments consist of *pinned* and *non-pinned* memory used as caches. Pinned memory is memory that can not be swapped; it is used to increase performance.

A cache known as the *pagepool* is stored in the pinned memory. It stores user data and metadata that can potentially improve I/O operation performance. For example, consider a program which can overlap I/O writes with computation. The program can write the data to the pagepool and quickly return control to the program without waiting for the data to be physically written to disk. Later GPFS can asynchronously write the data from the pagepool to disk while the CPU is crunching on other data. Similarly, when GPFS can detect a read pattern, it can asynchronously *prefetch* data from disk while the CPU is crunching some other data and place it in the pagepool so that by the time the application program reads the data, it only needs to fetch it from memory and not have to wait while the data is fetched from disk.

The size of this cache is controlled by the pagepool parameter of the **`mmconfig`** and **`mmchconfig`** commands. The size of the pagepool specified by this parameter is merely an upper limit of the actual size of the pagepool *on each node*. GPFS will dynamically adjust the actual size of the pagepool up to the value specified by the pagepool parameter to accommodate the current I/O profile. Currently, the maximum value of the pagepool parameter is 512MB while the minimum is 4 MB; the default is 20 MB (MB represents 1048576 bytes or one megabyte).

In addition to the pagepool, there is a non-pinned cache that stores information on opened and recently opened files. There are two classes of information stored in this memory; metadata (i.e., i-nodes) and stat()[1] information. They are called the *i-node cache* and *stat cache* respectively.

The size of the i-node cache is *controlled*, but not set, by the maxFilesToCache parameter in the **mmconfig** and **mmchconfig** commands. The actual number of i-nodes present in this cache is determined by how many times maxFilesToCache exceeds the number of files having information in this cache; if exceeded often, there will be fewer i-nodes stored in this cache than when it is seldom exceeded.

The stat cache is quite different. Each cache line contains only enough information to respond to a stat() call and is 128 bytes long. The number of entries reserved for this cache is a maxStatCache * maxFilesToCache where maxStatCache = 4 by default. **mmchconfig** is used to change this value.

This non-pinned cache is most useful in applications making numerous references to a common file over short durations, as is done for file systems containing user directories or in transaction processing systems. It is less helpful for long duration number crunching jobs where there are only a small number of files open and they remain open for long durations.

When discussing these various caches, the term cache is frequently used generically and collectively to refer to all three types of cache (i.e., the pinned pagepool and non-pinned i-node and stat caches), but it is also used to refer generically to the pagepool alone (since the pagepool is a cache). When it's important, the context makes the intent of the authors clear.

2.4.2 When is GPFS cache useful

Simple, definitive rules characterizing where the GPFS cache is effective and not effective are difficult to formulate. The utility of these various caches and the optimum size of the controlling parameters are heavily application dependent. But for applications that can benefit from them, setting their size too small can constrain I/O performance. Setting the value arbitrarily to its maximum may have no affect and may even be wasteful.

Regarding the pagepool specifically, a smaller pagepool size is generally most effective for applications which frequently reference a small set of records over a long duration (i.e., temporal locality). A larger pagepool is generally most effective for applications which process large amounts of data over shorter durations, but in a predictable pattern (i.e., spatial locality).

There are two occasions when these caches have no statistically measurable effect. The first is when the I/O access pattern is genuinely random and the file's user data can not be contained in its entirety within the pagepool. No access patterns can be predicted that allow GPFS to optimally schedule asynchronous

[1] The stat() function is used to retrieve file information such as size, permissions, group ID, etc. It is used by commands like **ls -l** and **du**.

transfers between disk and cache, and records do not reside in cache long enough to be re-used. The second situation occurs when the connections between disk and the CPU/memory bus are saturated. No amount of caching can compensate for such a heavy load being placed upon the inter-connections.

In the end, careful benchmarking using realistic workloads or synthetic benchmarks (see Appendix G, “Benchmark and Example Code” on page 237) that faithfully simulate actual workloads is needed to configure the GPFS caches optimally. However, this parameter can easily be changed using the **mmchconfig** command if necessary.

2.4.3 AIX caching versus GPFS caching: debunking a common myth

A common misunderstanding associated with GPFS is that AIX buffers data from the write() and read() system calls in AIX’s virtual memory (i.e., page space) before or after going through the pagepool. For instance, the authors have been counselled to be sure that the file size used in a GPFS benchmark significantly exceeds available node memory so that it is not artificially skewed by this buffering action of AIX. There is no need for AIX to buffer GPFS data since GPFS has exclusive use of its own private cache. The Example 2-3 illustrates this point.

Example 2-3 GPFS data is not buffered by AIX

```
host1t:/my_dir> ls -l
-rw-r--r--    1 root     sys        1073741824 Feb 12 18:28 file1
-rw-r--r--    1 root     sys        1073741824 Jan 29 21:42 file2
-rw-r--r--    1 root     sys        1073741824 Feb 24 13:02 file3
-rw-r--r--    1 root     sys        1073741824 Feb 17 19:17 file4
host1t:/my_dir> cp file1 /dev/null
host1t:/my_dir> time diff file1 file2
real    1m49.93s
user    0m41.83s
sys     1m7.89s
host1t:/my_dir> time diff file3 file4
real    1m48.00s
user    0m39.79s
sys     1m8.14s
```

Suppose a node has an 80 MB pagepool, 1512 MB of RAM, 1512 MB of page space and /my_dir is contained in a GPFS mounted file system. Notice that the files in Example 2-3 are significantly larger than the pagepool, and one of them can easily fit in the page space, but not several. Each file has identical contents. This test is done on an idle system. If AIX buffers the data as suggested, the **cp** command shown in Example 2-3 would indirectly place the contents of file1 in the page space as it is read and retain it for a little while. Then, when the **diff** command follows, file1 is referenced from the copy in memory (provided that it is

done before other tasks force that memory to be flushed) saving the overhead and time of reading file1 from disk again. Yet, when **`diff`** is executed the second time with different files not already cached in the page space, it takes nearly the same amount of time to execute! This observation is consistent with the design specifications for GPFS.

By contrast, a similar experiment (the files were only 256KB) conducted using JFS (which does buffer JFS file data in the AIX page space) showed that copying the file to /dev/null first allowed the **`diff`** operation to run nearly 3X faster; i.e., it makes a big difference in JFS. But *not* having this AIX buffering action in GPFS is *not* a loss; its just not needed. When a JFS file is not buffered, JFS actions go slower, but the GPFS actions always go faster as they are always cached (provided their I/O access pattern allows efficient caching). For instance, the un-buffered JFS action took 3X longer than either of the GPFS actions in the example above.

3

The cluster environment

This chapter focuses on the details of the implementation of GPFS in a cluster environment.

GPFS runs in various environments:

1. In an SP environment using VSD
2. In an SP environment using HACMP/ES and SSA disks or disk arrays instead of VSD
3. In a cluster environment using HACMP/ES and SSA disks or disk arrays

All implementations of GPFS rely on the IBM Reliable Scalable Cluster Technology, (RSCT), for the coordination of the daemon membership during the operation of GPFS and recovery actions in case of failure. The use of a quorum rule in conjunction with disk fencing ensures data integrity in failure situations.

HACMP/ES is a clustering solution for AIX, designed for high availability. It provides the operating and administrative environment for the subsystems of RSCT in a cluster environment. This chapter ends with an overview of HACMP/ES.

3.1 RSCT basics

In AIX, a *subsystem* is defined as a daemon that is under administration of the System Resource Controller, (SRC). A *distributed subsystem* is a distributed daemon under control of the SRC.

In this redbook, a cluster is defined as a set of RS/6000 hosts that share at least one network and are under the administration of a system of distributed daemons that provide a *clustering environment*.

For the systems we discuss in this book, the clustering environment is realized by the *IBM Reliable Scalable Cluster Technology, (RSCT)*. RSCT is a software layer that provides support for distributed applications. RSCT implements tasks that are commonly required by distributed applications, such as a reliable messaging service between daemons on different nodes and a mechanism for synchronization. Using the services provided by RSCT reduces the complexity of the implementation of a distributed application. RSCT can support multiple distributed applications simultaneously.

RSCT, a component of the IBM Parallel Systems Support (PSSP) software, consists of the following three distributed subsystems:

1. Topology Services (TS)
2. Group Services (GS)
3. Event Management (EM)

Figure 3-1 on page 29 shows the client server relationship between the three subsystems.

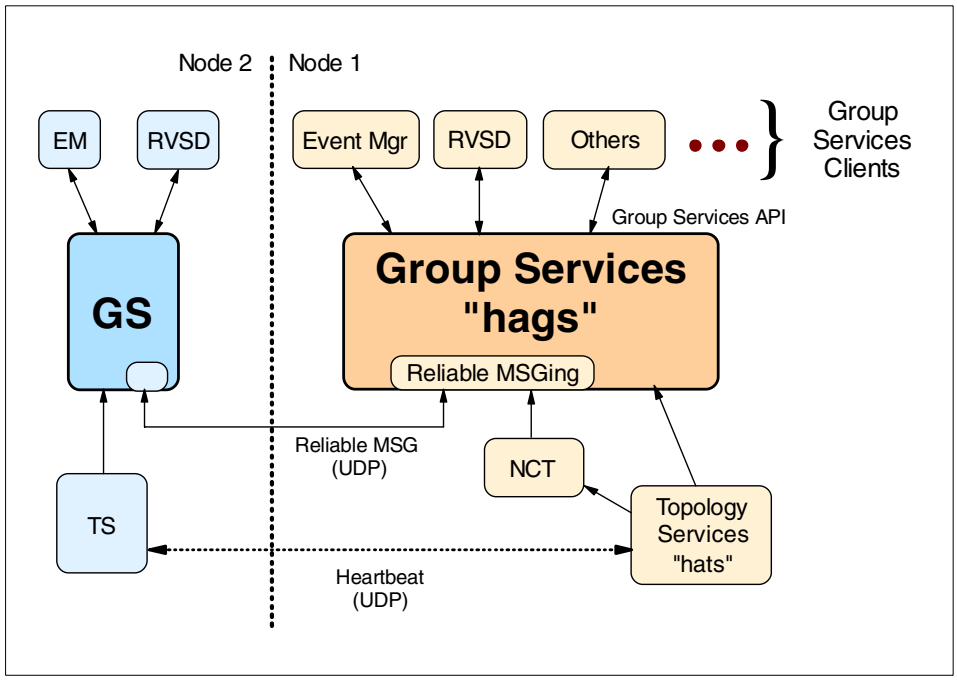


Figure 3-1 RSCT

RSCT provides applications with its services for a certain scope. An application may consist of multiple processes that run on multiple RS/6000 machines. Therefore, when the application uses the services provided by the RSCT, it must consider boundaries in which the application can use them.

An *RSCT domain* is the collection of nodes (SP node or an RS/6000 machine running AIX) on which the RSCT is executing. There are two types of domains for RSCT:

- SP domain
- HACMP domain

An SP domain includes a set of SP nodes within an SP partition. However, an HACMP domain includes a set of SP nodes or non-SP nodes defined as an HACMP/ES cluster.

A domain may not be exclusive. A node may be contained in multiple domains. Each domain has its own instance of RSCT daemons. Hence multiple instances of a daemon can be active on a node and each instance having separate configuration data, log files, etc.

3.1.1 Topology Services

Topology Services monitors the networks in the RSCT domain. Nodes in the RSCT domain are connected by one or more networks. For a network to be monitored by Topology Services, it has to be included into the configuration data for Topology Services. Topology Services, as distributed subsystem, relies on all daemons having the same configuration data.

Topology Services provides the following information to its clients:

State of adapters
Adapters are monitored by keepalive signals. If an adapter is detected as unreachable (e.g., due to a hardware failure), it will be marked as down.

State of nodes
The state of nodes is deduced from the state of adapters. If no adapter on a node is reachable, then the node is assumed to be down.

Reliable messaging library
The reliable messaging library contains a representation the network in the RSCT domain in form of a connectivity graph. It resides in a shared memory segment. Clients can use the reliable message library to determine communication paths for the message passing between the daemons on distinct nodes. If the state of an adapter changes, the reliable messaging library is updated.

Clients of Topology Services, can subscribe to be updated about the status of adapters, and typically use this information for the implementation of error recovery.

3.1.2 Group Services

Group Services provides an infrastructure for the communication and synchronization among the peer daemons of a distributed system. The daemons of a distributed system, to cooperatively act on their domain, need to pass information between them and perform tasks in a coordinated way, that often requires a synchronization among them at intermediate steps.

The implementation of a synchronization algorithm in any distributed system is very complex, especially since error recovery (e.g., when a node fails during the execution of a synchronization algorithm) is commonly a requirement.

Group Services provides algorithms for the synchronization and information sharing between the daemons of distributed systems, that abstract from the specifics of the tasks, performed by any single distributed system, for which the synchronization is required. By using Group Services, the implementation of a distributed system becomes less complex and less expensive.

The daemons of distributed systems that use Group Services connect to it locally on each node as clients. The synchronization algorithm is referred to as *Group Services voting protocol*. Clients of Group Services are members of one or more Group Services *groups*. All members of a group participate in the same voting protocols. A member of a group can be a *provider*, a *subscriber* for that group, or both. Daemons of different distributed systems can belong to the same group, hence Group Services facilitates the interaction between different distributed systems.

A Group Services voting protocol consists of one or more phases, they are called one and n-phase voting protocols. A phase in the voting protocol entails the distribution of information to all active members of the group and the acknowledgement of receipt by all members. A one-phase voting protocol is simply a distribution of information by one member to all others with acknowledgement of receipt of that information. In an n-phase voting protocol, any member can propose a next phase of voting at the completion of each phase, depending on the outcome of the actions that are performed by that daemon locally on the node in context with that phase. The phases of a voting protocol are performed throughout the group in a serial way; a new phase is started only if all members have finished the current one. Different phases of an n-phase protocol are separated by *barriers*, the daemons on all nodes have to reach a barrier before the next phase can be started. This is how Group Services archives synchronization between daemons on distinct nodes.

Group Services maintains groups internally, to distribute to subscribers the state of adapters, and nodes. Topology Services is a provider for this group. The Group Services daemons on different nodes communicate with each other using the sockets established by Topology Services.The reliable messaging library that is maintained by Topology Services is used to determine a valid route. Clients access Group Services through the Group Services API (GSAPI).

GPFS and HACMP/ES are clients of Group Services and use it for the synchronization of recovery actions after failures and for the coordination of the membership of daemons.

For more details about Group Services the reader is referred to *RSCT: Group Services Programming Guide and Reference,* SA22-7355, and *RSCT Group Servcies: Programming Cluster Applications,* SG24-5523.

3.1.3 Event Management

The Event Management subsystem monitors system resources on the nodes in the RSCT domain and provides its clients with the state of these resources. System resources are processes, or hardware components with AIX configuration settings pertaining to them. System resources are monitored by *Resource Monitors*. A Resource Monitor is a client of the Event Management daemon, connected to it by the Resource Monitor Application Program Interface (RMAPI).

Clients of Event Management that are to be informed about the state of a resource need to register for notification about that resource.

The daemons of the Event Management subsystem communicate among each other using Group Services; Event Management is a client of Group Services. The daemons of the Event Management subsystem use Group Services for the information sharing.

The cluster manager subsystem of HACMP/ES is a client of Event Management. For more details about the Event Management subsystem the reader is referred to *RSCT: Event Management Programming Guide and Reference,* SA22-7355.

3.2 Operating environments for GPFS

GPFS is supported in three different operating environments.

GPFS on the SP

On an SP, GPFS exists in two environments that are distinguished by the requirement of presence of the Virtual Shared Disk (VSD) layer.

1. VSD environment
   - the RSCT component of PSSP
   - the VSD and RVSD component of PSSP
   - bandwidth of the high speed SP switch network
   - SP infrastructure, for the configuration and administration of the RSCT domain.
2. non-VSD environment
   - the RSCT component of PSSP
   - disk architecture that provides local access of each node to all disks
   - HACMP/ES for the configuration and administration of the RSCT domain in a cluster environment

GPFS in a cluster of RS/6000 nodes

In a cluster of RS/6000 nodes, GPFS is supported in a cluster environment.

3. cluster environment
   - RSCT, as part of HACMP/ES
   - disk architecture, that provides local access of each node to all disks
   - HACMP/ES for the configuration and administration of the RSCT domain in a cluster environment

Note that all three implementations of GPFS rely on a clustering environment, that is provided by RSCT.

3.2.1 GPFS in a VSD environment

The implementation of GPFS in a VSD environment relies on the IBM Virtual Shared Disk (VSD), and on the IBM Recoverable Virtual Shared Disk (RVSD), component of PSSP. It is only supported on the SP, since it requires the bandwidth of the high speed network of the SP. GPFS and the RVSD subsystem requires RSCT to provide a clustering environment.

VSD subsystem

The Virtual Shared Disk subsystem provides uniform disk access for all nodes in its domain to raw logical volumes that are configured on disks under its administration. Each disk is locally connected to at least one node in the domain, but not required to be locally connected to all nodes. The logical volumes that are managed by the VSD subsystem appear on all nodes in the VSD domain as virtual shared disks. Applications access virtual shared disks on all nodes like raw logical volumes.

Anode (to which a disk is locally connected) serves as a VSD server for each disk. I/O requests to a virtual shared disk (on any node) are forwarded to the VDS server of the disk on which the corresponding logical volume resides. The I/O traffic is routed over the high speed embedded network of the SP.

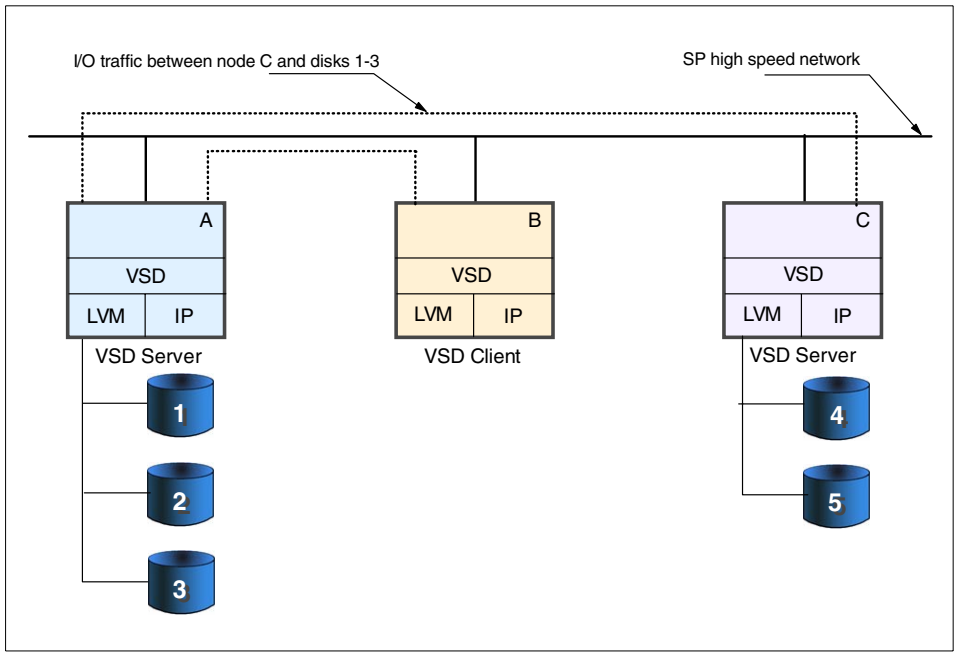


Figure 3-2 VSD environment

Figure 3-2 shows a cluster of three nodes. I/O traffic between node C and disks 1-3 is routed over host A, which is the VSD server for those disks. In regards to I/O operations for disks 1-3 Node A is called the server node, and nodes B, and C are called client nodes.

RVSD subsystem

The RVSD subsystem provides high availability for the virtual shared disk. If a disk is locally connected to more than one node, two nodes can be configured to act as VSD servers. They are referred to as *primary* and *secondary* VSD servers. By default, the primary VSD server will be the server for that disk. If a failure affects the primary VSD server, such as a loss of network connectivity, a disk adapter or node failure, or a failure of the VSD server itself, the secondary VSD server will take over, and all I/O requests will be routed to it instead. Once the failure on the primary VSD server has been has been resolved, the primary node will resume it's role as the VSD server.

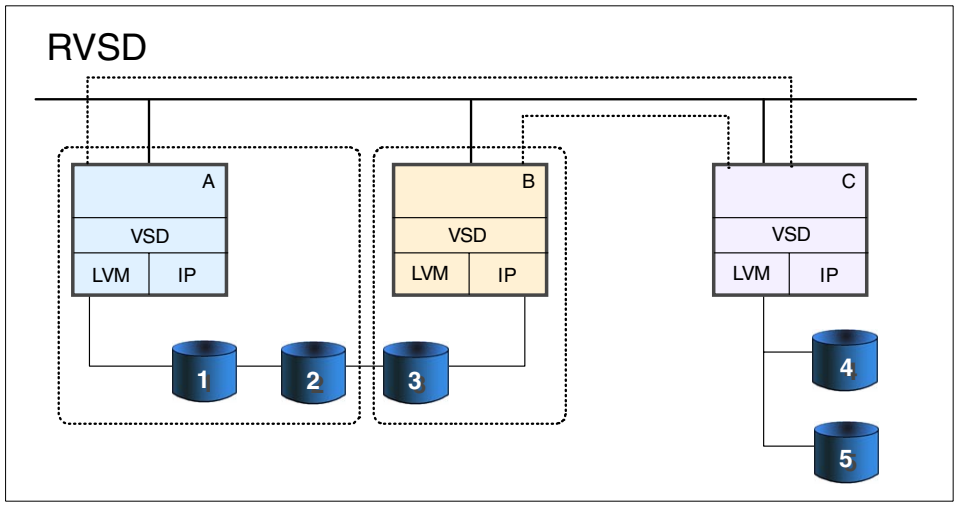


Figure 3-3 RVSD environment

Figure 3-3 shows a VSD configuration, and a mutual takeover situation for the VSD server of disks 1-3. Disks 1 and 2 have node A as the primary VSD server, and node B as the secondary VSD server; disk 3 has node B as the primary and node A as the secondary VSD server.

Disk fencing is performed in failure scenarios. GPFS uses the capability of the RVSD for disk fencing. Therefore the RVSD subsystem is a requirement for the implementation of GPFS in a VSD-environment. The role of disk fencing for error recovery in GPGS is explained in more detail in Section 3.3.5, “Disk fencing” on page 43.

The reader may expect that the Concurrent Logical Volume manager is required by the RVSD to allow concurrent access of the primary and secondary VDSM to the disks in order to reduce fallover times. However, this is not the case; the CLVM is not required. The volume groups in a VSD environment are not created as concurrent capable, they are varied on at only one node. If the secondary VSD server takes control, it varies on the volume group, perhaps breaking the disk reserve. The vary on only takes about ten seconds; the GPFS daemon can tolerate time-outs for disk access up to 30 seconds.

For more details about VSD and RVDS, see *PSSP 3.2: Managing Shared Disks,* SA22-7349.

3.2.2 GPFS in a non-VSD environment

GPFS requires all nodes to have local access to all disks under its administration. Local access of multiple nodes of the same set of disks is inherent to certain disk architectures, for instance SSA or Storage Area Networks. The implementation of GPFS in a non-VSD environment relies on local access of all nodes of the GPFS domain to all disks by the nature of the disk architecture. Since all nodes have local access to all logical volumes, no routing of I/O traffic over an external network is required, as it is in the VSD environment.

SSA

Currently, in version 1.4 of GPFS, Serial Storage Architecture (SSA) is the only disk technology that is supported in a non-VSD environment. This limits the number of nodes to eight, since an SSA loop cannot contain more than eight host adapters. Hence, at maximum, eight nodes can have direct access to any disk. However, nodes can participate in more than one SSA loop. In order to use SSA disks for GPFS, a concurrent capable volume group should be configured on each disk with each one containing a logical volume.

3.2.3 GPFS in a cluster environment

The implementation of GPFS in a cluster environment is similar to the implementation of the cluster in a VSD environment with the difference being that each node in the cluster environment must have access to the directly attached disks. Currently, in version 1.4 of GPFS, SSA is the only supported disk architecture in the cluster environment.

RSCT and HACMP/ES build the clustering environment for GPFS outside an SP in a cluster of RS/6000 nodes. HACMP/ES provides the means for the configuration and administration of the RSCT domain, which is given by the cluster topology of HACMP/ES (see Chapter 3.4, “Implementation details of GPFS in a cluster” on page 45). HACMP/ES requires redundant networking connections between all nodes.

The HACMP/ES cluster constitutes an RSCT domain. Throughout this chapter, the RSCT domain will be referred to as the HACMP/ES cluster.

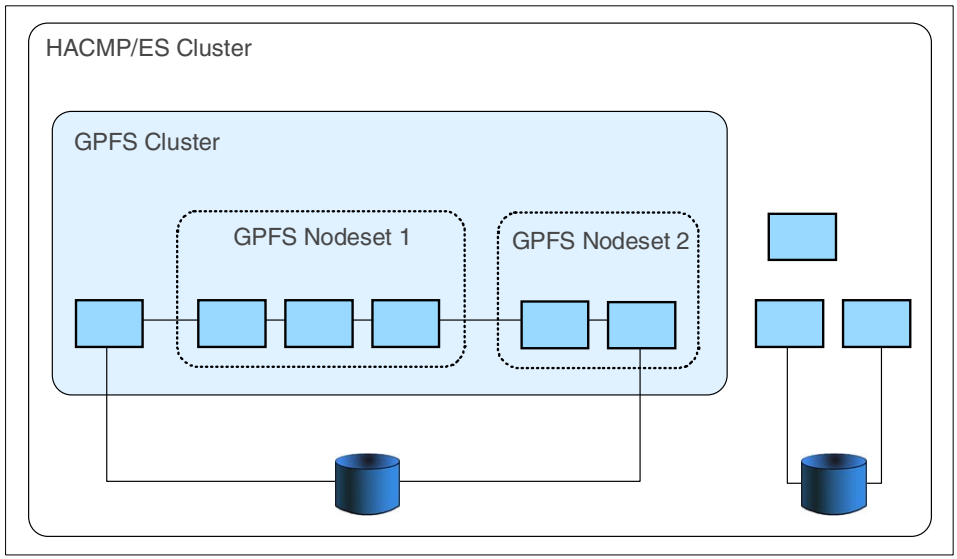


Figure 3-4 Clustering domains and GPFS nodesets

Figure 3-4 shows the relationship between the clustering domains, and GPFS nodesets. All nodes that belong to a GPFS cluster must belong to the same HACMP/ES cluster. Only one GPFS cluster can be configured within one HACMP/ES cluster, which may contain further nodes that are not part of the GPFS cluster. A GPFS cluster can contain multiple GPFS nodesets; nodes can be dynamically added to, or removed from, a nodeset. GPFS nodesets are disjoint, thus, a node cannot belong to two nodesets. All nodes that will be used in the GPFS cluster need to have direct access to all disks.

SSA currently is the only disk architecture supported for GPFS in a cluster environment, which limits the number of nodes in a GPFS cluster to eight.

The setup of GPFS in a cluster entails the configuration of the HACMP/ES cluster topology, which is simple and straightforward from an operational point of view.

3.3 GPFS daemon state and Group Services

Most of the synchronization in the GPFS subsystem is implemented in the multithreaded model of the GPFS distributed daemon itself. During normal operation, the GPFS daemons communicate with each other using TCP/IP socket connections. Topology Services monitors the network adapters, on which the sockets are configured, for failures.

Group Services are used to synchronize all actions required to bring a GPFS daemon into an active state and for the handling of failures. If a failure has been detected, the affected GPFS daemon leaves the active state and recovery actions are performed to protect the integrity of the file system and the GPFS subsystem.

Group Services maintain two groups for each GPFS nodeset, Gpfs.name and GpfsRec.name, whereby .name is the name of the nodeset. This is an implementation detail due to the architecture of Group Services (see Section 3.3.3, “Coordination of event processing” on page 41).

3.3.1 States of the GPFS daemon

A GPFS daemon has three states, *down*, *initializing*, and *active*.

down

The GPFS daemon is down if it is shown inoperative by the System Resource controller as shown in Example 3-1.

Example 3-1 GPFS daemon in the down state

```
host1t:/> lssrc -s mmfs
Subsystem         Group            PID     Status
 mmfs             aixmm                    inoperative
```

initializing

In Example 3-2, the GPFS daemon has been started by the **mmstartup** command, which is a Shell script, to start the GPFS daemon on one or more nodes. It issues a start of the daemon by the System Resource Controller on each specified node. Once **mmstartup** has completed, the GPFS subsystem is shown active by **lssrc**. The system resource controller will issue the **runmmfs** command, which is another Shell script.

Example 3-2 GPFS daemon in the initializing state

```
host1t:/> mmstartup
Sat Mar 10 00:19:14 EST 2001: mmstartup: Starting GPFS ...
0513-059 The mmfs Subsystem has been started. Subsystem PID is 18128.
host1t:/> lssrc -s mmfs
Subsystem         Group            PID     Status
 mmfs             aixmm            18128   active
host1t:/> ps -ef | grep mm
root 18130  8270   0 00:21:48      -  0:00 ksh /usr/lpp/mmfs/bin/runmmfs
```

To become part of the GPFS distributed subsystem, the GPFS daemon needs to connect to the Group Services daemon as a client. The script /usr/lpp/mmfs/bin/runmmfs tries periodically to connect to Group Services as a client. This state is called the initialization state of the GPFS daemon. The attempt to join the GPFS groups may not be successful if Group Services is not active or the already active daemons in the GPFS groups deny a join.The latter would occur, for instance, if the network adapter on which the GPFS socket connection is established has been detected as failed by Topology Services.

active

If the GPFS daemon has connected with Group Services and joined the GPFS distributed subsystem, it is in the active state, as shown in Example 3-3, and ready to perform cooperatively with the other GPFS daemons that have connected with Group Services. It is shown as a member of the Groups that Group Services maintains for the nodeset. The script runmmfs will spawn the mmfs daemon.

Example 3-3 GPFS daemon in the active state

```
host1t:/> lssrc -ls grpsvcs
Subsystem         Group            PID     Status
 grpsvcs          grpsvcs          31532   active
4 locally-connected clients.  Their PIDs:
27790(hagsglsmd) 36584(mmfsd) 26338(haemd) 27234(clstrmgr)
HA Group Services domain information:
Domain established by node 3
Number of groups known locally: 5
                   Number of         Number of local
Group name         providers      providers/subscribers
Gpfs.set1              3              1          0
GpfsRec.set1           3              1          0
ha_em_peers            3              1          0
CLRESMGRD_111          3              1          0
CLSTRMGR_111           3              1          0
host1t:/> ps -ef | grep mm
root 19888  4994   0 21:12:03      -  0:01 /usr/lpp/mmfs/bin/mmfsd
```

If a failure that affects the GPFS daemon is detected by RSCT, the GPFS daemon will leave the active state and return to the initialization state.

3.3.2 The role of RSCT for GPFS

For a GPFS daemon to become a member or leave the active configuration, this entails state changes that are performed on all active nodes. They are performed in an n-phase voting protocol.

Further, Topology Services monitors the state of network adapters and publishes this information to the GPFS groups. A change is the state of a network adapter, which affects the functionality of the cluster, will result in a change of membership for the GPFS daemon on that node.

The role of RSCT in the implementation of GPFS is to detect and synchronize the necessary changes of all active GPFS daemons that pertain to the following:

Membership of GPFS daemons

When a GPFS daemon joins or leaves an active configuration, this entails multiple steps. All active GPFS daemons go through a multiple state change in order to perform the changes that are required in the entire active GPFS subsystem. This may entail the selection of new management nodes, the mounting or unmounting of the file system, and an update of the fence registers of disks. Those changes are synchronized by Group Services in a multiple phase protocol.

State of the network adapters that support the GPFS communication

The GPFS daemons on all nodes subscribe to the adapter group that is updated by Topology Services with the state of the adapters. If the adapter that carries the socket for the GPFS communication has failed on a node, the GPFS daemon on that node can no longer communicate with the other GPFS daemons. Upon detection of the failure of the network adapter that carries the socket for GPFS, the GPFS daemon on that node will leave the GPFS groups and transition into the initialization state.

Connectivity to a node in the GPFS cluster

There are several reasons why a node, which had previously been detected by Topology Services, may not be detected. The node itself may have failed or network connectivity to it may be lost. Upon detection of the loss of connectivity to any node on which GPFS has been active, the remaining GPFS daemons will exclude that node from the GPFS groups to which they belong and perform recovery actions, such as disk fencing.

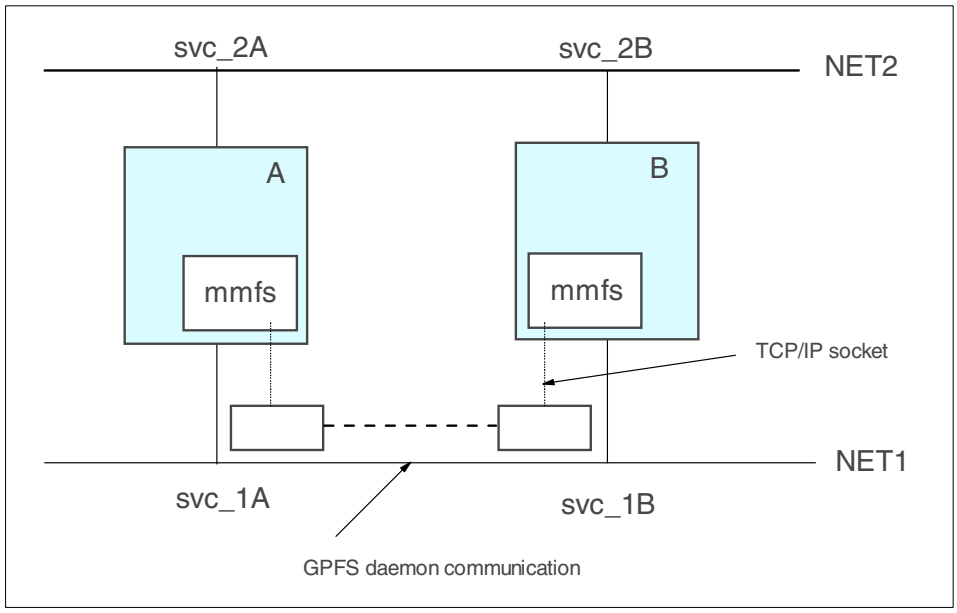


Figure 3-5 Error recovery by RSCT for a GPFS cluster

Figure 3-5 shows two nodes of a GPFS cluster. The two networks, NET1 and NET2, are configured to the part of the HACMP/ES cluster topology. Hence, they are monitored by Topology Services. The GPFS daemons between both nodes communicate with each other using TCP/IP socket connections that are established on the network devices of adapters, svc_1A and svc_2A. The members in the GPFS groups are informed about the state of the adapters svc_1A and svc_1B, the connectivity to nodes A and B, and the membership of the mmfs daemons in the GPFS groups.

The recovery actions performed by the GPFS distributed subsystem are the same that are performed upon the loss of a network adapter used for the GPFS communication or entire loss of connectivity to a node. However, redundant networking connections that are known to Topology Services are required, in an implementation that relies on HACMP/ES to maintain GPFS performance after failure (see Section 3.5.3, “Partitioned clusters” on page 51).

3.3.3 Coordination of event processing

Group Services maintains two groups for the GPFS subsystem to facilitate the processing of multiple failures simultaneously and to process daemon membership changes hierarchically.

Simultaneous processing of multiple failures

Group Services enforces the serial processing of recovery actions within one group. In a multiphase voting protocol, a new phase is started when all members have completed the previous phase and reached a barrier. Protocols within the same group are processed serially as well, a new protocol can only be started if no other protocol is currently processed.

If a failure of a GPFS daemon has been detected, the active GPFS daemons will start running a multiphase voting protocol in the group Gpfs.name to perform recovery actions. If another failure is detected, it will cause a protocol to be run in the group GpfsRec.name. This protocol will cause a termination of the protocol that is run in the group Gpfs.name and a restart of it with the updated information about the set of GPFS daemons that are affected by failures. The performance of recovery actions that are run during the n-phase protocol may take a while, since they involve an update of disk fence registers, etc. The actions that are performed in context of the protocol that is run in the second group, GpfsRec.name, do not take much time, thus multiple failures are effectively processed simultaneously.

Hierarchy of processing for join and fail requests

The n-phase protocols for join requests of GPFS daemons are run within the group Gpfs.name. A failure that is detected while a join request is run will cause a protocol to be run in the group GpfsRec.name, which causes an abort of the processing of the join request and the processing of the failure. The join request will be resubmitted periodically and the protocol will be started again after the protocol to process the failure has finished.

3.3.4 Quorum

Quorum is a simple rule to ensure the integrity of a distributed system and the resources under its administration. In GPFS, quorum is enforced.

The notion of quorum applies to the GPFS nodeset. In a GPFS nodeset, quorum is achieved if half (in a nodeset consisting of two nodes), or more than half (in a nodeset consisting of more than two nodes), of the GPFS daemons are in the active state (as described in Section 3.3.1, “States of the GPFS daemon” on page 38) and are able to communicate with each other. The latter may not be the case after multiple network failures that result in a partitioned configuration (see Section 3.5.3, “Partitioned clusters” on page 51).

In other words, quorum is achieved if more than half of the GPFS daemons are members of the same instance of a GPFS group. File systems that are configured in a GPFS nodeset can only be mounted if quorum is achieved.

For a nodeset consisting of two nodes, quorum is user selectable. If Single Node Quorum is not enabled, the GPFS daemons on both nodes have to be active for all actions that otherwise depend on quorum to succeed.

3.3.5 Disk fencing

Disk fencing provides the means to set access permissions for hosts that all are locally connected to a disk. The fence register of a disk is a hidden sector that contains the access permissions for all nodes that have access to the disk, which in a GPFS configuration are all nodes of the GPFS cluster.

In GPFS, disk fencing applies to a GPFS nodeset and all disks that are under its administration. A disk configured for GPFS is only accessible by the nodes of the nodeset to which it belongs.

Disk fencing is implemented for SSA technology in hardware and the device driver level. The status of fence registers is not accessible through AIX commands. In GPFS, the command **mmshow_fence** displays the content of the fence registers on all disks that belong to the GPFS nodeset from which the command was issued.

Disk fencing is implemented in GPFS according to a simple rule:

If quorum is achieved and the file system is mounted on any node in the GPFS nodeset, then all nodes on which the file system is not mounted are fenced out.

The set of disks that are fenced out varies during the runtime of GPFS.

3.3.6 Election of the GPFS global management nodes

The global management nodes are the configuration manager, the file system manager, and the metadata manager node. The latter exists for any open file and is assigned dynamically. The configuration manager exists for each GPFS nodeset and the file system manager for each file system in a GPFS nodeset. All global management functions are performed by separate threads within the GPFS daemon. See Section 2.2, “Global management functions” on page 16 for more information.

If a GPFS daemon leaves the active GPFS subsystem, the election of any of the above may be necessary. This is part of the recovery actions that are performed during the n-phase protocol that is run to change the daemon membership. First, if the daemon that leaves the active GPFS subsystem was the configuration manager, a new one will be elected. If quorum persists for the GPFS nodeset,

the configuration manager will elect a new file system manager for any file system for which the daemon that has left the active GPFS subsystem had been manager. The file system manager will initiate recovery actions for the file system metadata and elect new metanodes for ope files, if necessary.

Configuration manager

The configuration manager is assigned internally. It does not impose any significant load onto the system. The first GPFS daemon in a GPFS nodeset that become active will be the configuration manager. It it leaves the active state, the GPFS daemon with the next lowest node number that is active will be elected as the new configuration manager.

Which node acts as the configuration manager, is normally not of interest from an operational point of view. However, it can be determined by the command **`mmfsadm dump cfgmgr`**.

File system manager

The file system manager is elected by the configuration manager after it has been determined that quorum exists. If quorum does not persist, all file systems will be unmounted and file system managers are not required. The file system manager handles all token traffic and the assignment of metanodes, hence it can impose an additional load onto the system.

A node can be specified not to perform as file system manager when creating the nodeset via **`mmconfig`** or when modifying the configuration using **`mmaddnode`**, or **`mmchconfig`**. There must be at least one node in the nodeset eligible for file system management. However, GPFS will not always honor the user specified exclusion of a node from file system management functions. For example, if all nodes are excluded, then GPFS will elect one of the excluded nodes.

The command **`mmlsmgr`** lists all nodes that currently perform as file system managers for the file systems in the GPFS nodeset. The command **`mmfsadm dump cfgmgr`** lists all active file system managers as well.

Metanode

Metanodes are assigned internally. Usually the node which has the file open for the longest amount of time is the Metanode for that file.

3.4 Implementation details of GPFS in a cluster

HACMP/ES is designed to provide high availability for applications. An application is highly available if it is restored within a short time span of a failure. HACMP/ES monitors the application and system resources on which it relies and performs recovery actions if a failure has been detected. The recovery actions are coordinated by the HACMP/ES Cluster Manager, which is a distributed daemon that uses Group Services for synchronization.

The implementation of GPFS for HACMP/ES does not make use of the capabilities of HACMP/ES for high availability. HACMP/ES provides the operating environment that GPFS requires for the subsystems of RSCT.

The HACMP/ES cluster services refer to the set of all distributed subsystems that are part of HACMP/ES. The following subsystems are always activated when the HACMP/ES cluster services are started on a node:

Topology Services
Group Services
Event Management
HACMP/ES Cluster Manager
HACMP/ES SMUX Peer Daemon

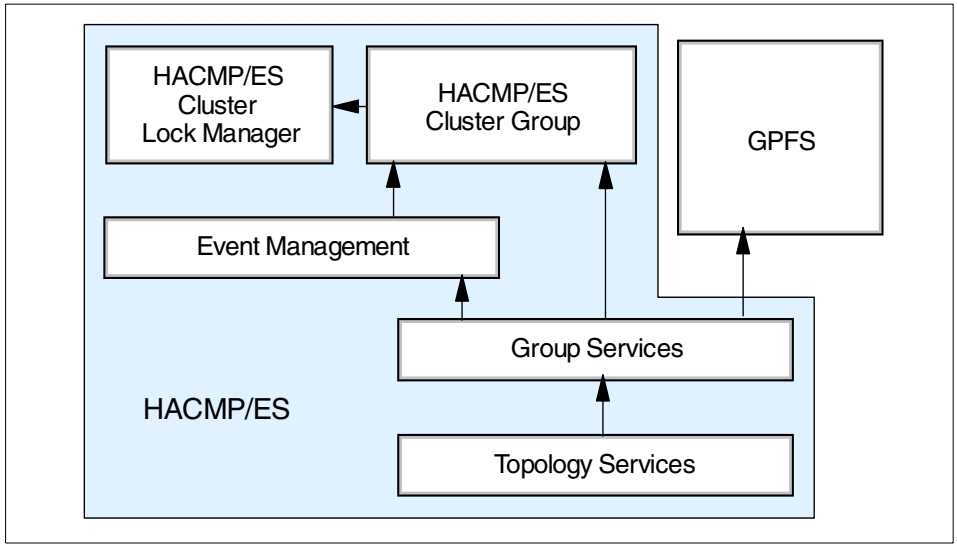


Figure 3-6 GPFS in the HACMP/ES operating environment

Figure 3-6 on page 45 shows the client server relationship of all active subsystems in the implementation of GPFS in a cluster. The HACMP/ES cluster group contains the following daemons:

HACMP/ES Cluster Manager
HACMP/ES SMUX Peer Daemon
HACMP/ES Cluster Information Daemon

GPFS, Event Management, and the HACMP/ES Cluster Manager are clients of Group Services. There is no interaction between the subsystems of the HACMP/ES cluster group and the GPFS subsystem. The HACMP/ES Cluster Lock Manager is a client of the HACMP/ES Cluster Manager.

HACMP/ES provides the means to configure and administer the operating domain for RSCT. In particular:

- Configuration of the HACMP/ES cluster that defines an RSCT domain, which can be changed dynamically, i.e. while the daemons are active
- Environment to start and stop the RSCT subsystem
- Cluster monitoring tools

An HACMP/ES cluster, while providing an operating domain for GPFS, can be used to make other applications highly available. Few restrictions apply; see Section 3.6.1, “Configuring HACMP/ES” on page 56 for more information.

3.4.1 Configuration of the cluster topology

The cluster topology of an HACMP/ES cluster consists of:

nodes
The set of nodes is the operating domain for HACMP/ES and RSCT.

adapters
Adapters are monitored by Topology Services by keepalive signals and used for the communication within the cluster. They can be used to make IP addresses highly available (see Section 3.6.1, “Configuring HACMP/ES” on page 56).

networks
Networks define the association of adapters to physical networks; two adapters that belong to the same cluster network also belong to the same physical network.

tuning parameters
Tuning parameters describe the frequency of keepalive signals, which are sent by Topology Services, and grace periods, which designates the maximum time span keepalive signals can be missed without issuing a failure notification.

An HACMP/ES cluster is required to have redundant networking connections between nodes to provide recovery for failures and typically have multiple network adapters that connect a node to any network.

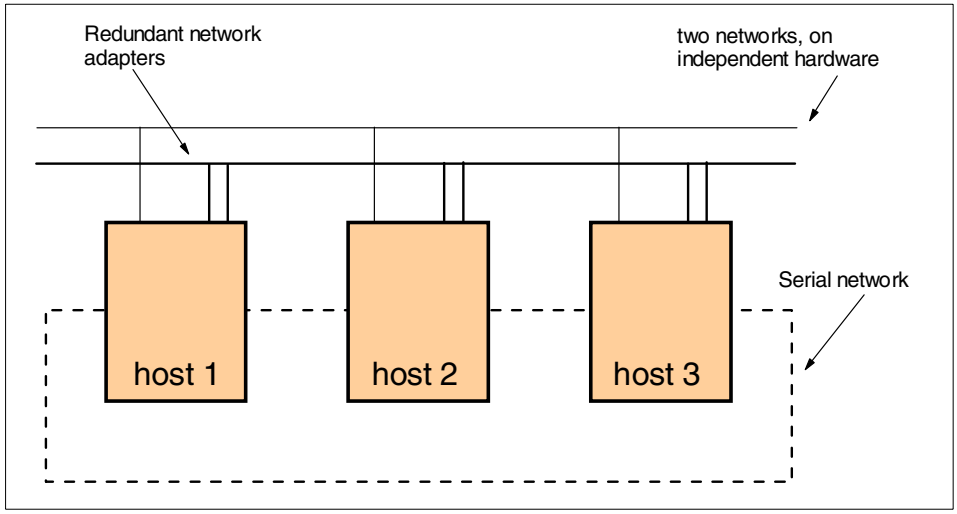


Figure 3-7 HACMP/ES cluster topology

Figure 3-7 shows a cluster of three nodes that all are connected by two TCP/IP networks and a serial network. All nodes and networks are configured in the HACMP/ES cluster topology, therefore they will be monitored by Topology Services and used for message passing. Two networks, which are configured on independent hardware components, ensure fault tolerance for the network connectivity between nodes. One network has two host adapter connections with each node.

3.4.2 Starting and stopping the subsystems of RSCT

The three states that a GPFS daemon can assume are explained in Section 3.3.1, “States of the GPFS daemon” on page 38. The start of the HACMP/ES cluster services entails starting the subsystems of RSCT.

Example 3-4 GPFS daemon is initializing; HACMP/ES cluster is down

```
host1t:/> ps -ef | grep mm
root 30012  6204   0 01:37:13      -  0:00 ksh /usr/lpp/mmfs/bin/runmmfs
host1t:/> lssrc -s mmfs
Subsystem         Group            PID Status
 mmfs         aixmm            30012   active
host1t:/> lssrc -s clstrmgrES
Subsystem         Group            PID Status
clstrmgrES    cluster           inoperative
```

If the GPFS subsystem has been started on a node while the cluster services are still inactive, the GPFS will be in the initializing state, as shown in Example 3-4. After the cluster services have been started and Group Services has become active, the GPFS daemon will attempt to join the GPFS groups and transition into the active state. Furthermore, the HACMP cluster manager will connect with Group Services after it is started.

Example 3-5 GPFS and HACMP/ES are both active

```
host1t:/> ps -ef | grep mm
    root 30012  6204   0 01:37:13      -  0:01 /usr/lpp/mmfs/bin/mmfsd

host1t:/> lssrc -ls grpsvcs
Subsystem         Group            PID     Status
 grpsvcs          grpsvcs          29606   active
4 locally-connected clients.  Their PIDs:
15444(hagsglsmd) 30012(mmfsd) 18412(haemd) 20498(clstrmgr)
HA Group Services domain information:
Domain established by node 3
Number of groups known locally: 5
                  Number of     Number of local
Group name         providers    providers/subscribers
Gpfs.set2          4               1          0
GpfsRec.set2       4               1           0
ha_em_peers              4           1           0
CLRESMGRD_111      4               1           0
CLSTRMGR_111       4               1           0
```

Example 3-5 shows the groups maintained by Group Services when GPFS and HACMP/ES are active. Group Services maintains five different groups on host1. The groups Gpfs.set2, and GpfsRec.set2 correspond to the GPFS nodeset, to which this node belongs, which has the nodeset name set2. The groups CLRESMGRD_111, and CLSTRMGR_111 are maintained for the HACMP/ES Cluster Manager subsystem. The HACMP/ES cluster ID is 111.

Stopping the HACMP/ES cluster services entails stopping the subsystems of RSCT as well. The GPFS daemon, if it has been in the active state, will transition into the initializing state once Group Services has become inactive.

3.4.3 Dynamic reconfiguration of the HACMP/ES cluster topology

The HACMP/ES cluster topology can be changed dynamically, i.e. while the cluster services are active on some nodes of the cluster. This is called Dynamic Automatic Reconfiguration Event (DARE). Most changes in the HACMP/ES cluster topology can be performed dynamically, in particular, adding or deleting nodes and network adapters.

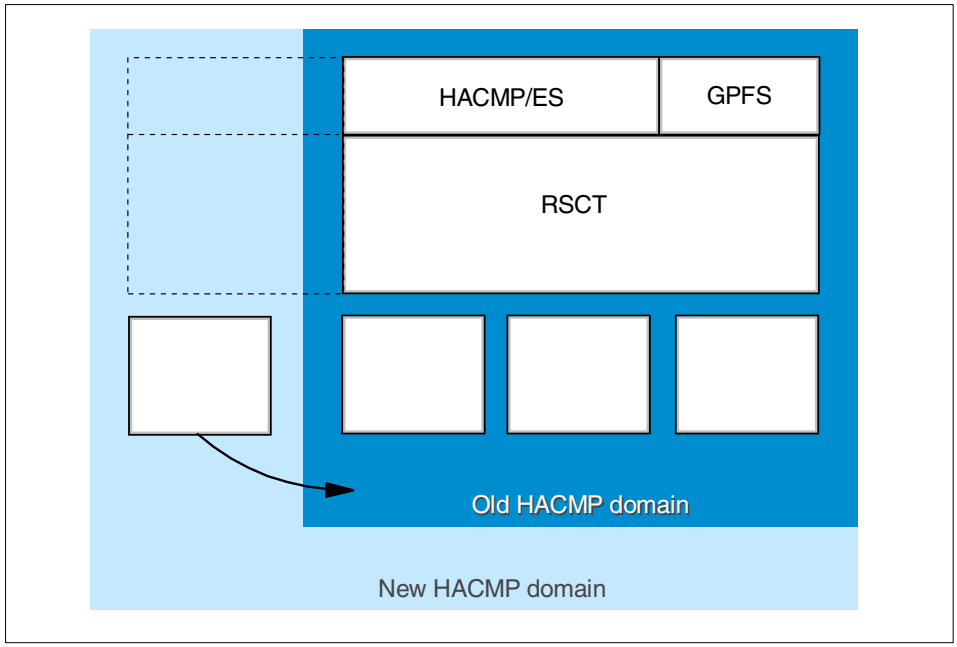


Figure 3-8 Adding a node the HACMP/ES cluster dynamically

Figure 3-8 illustrates a dynamic reconfiguration to extend the HACMP domain, which consists of three nodes. A new node is added while the HACMP/ES cluster services and GPFS are active on all nodes. After the node has been added, the HACMP/ES cluster services can be started. After a node is properly configured into the HACMP cluster, it can be added to the GPFS configuration.

3.5 Behavior of GPFS in failure scenarios

This section describes the recovery actions of GPFS in some failure scenarios.

3.5.1 Failure of an adapter

Figure 3-9 illustrates the behavior of a GPFS daemon when it attempts to join the active GPFS subsystem while the adapter that supports the GPFS socket is detected as failed.

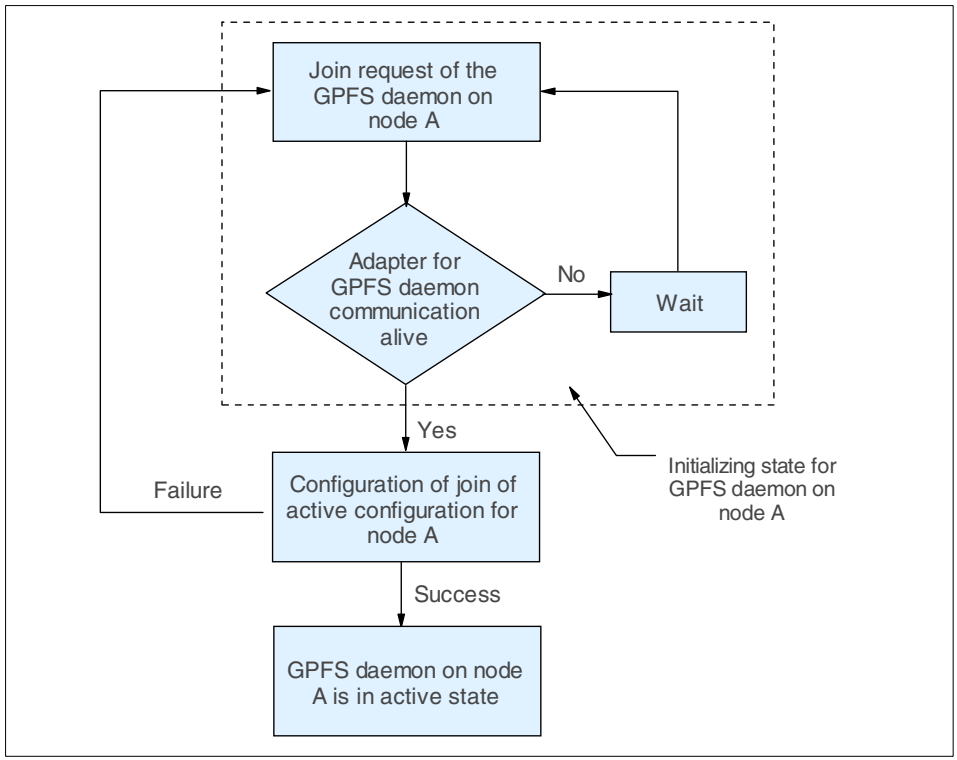


Figure 3-9 Join attempt of a GPFS daemon

3.5.2 Failure of a GPFS daemon

Figure 3-10 illustrates the behavior of GPFS when an active GPFS daemon fails on a node. The same sequence of recovery actions is run if the adapter of the network dedicated to GPFS has failed.

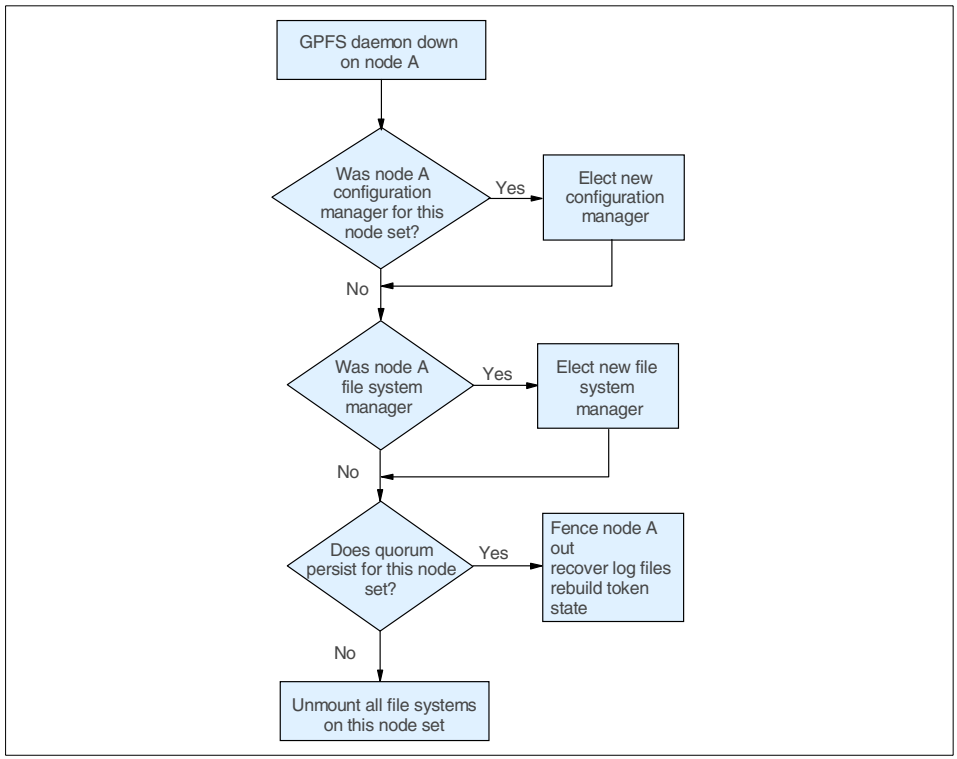


Figure 3-10 Failure of a GPFS daemon

3.5.3 Partitioned clusters

Due to multiple network failures, a cluster can subdivided into two or more sets of nodes that can communicate with each other but not with nodes that belong to another set. The nodes within such a set form a *partition*; the cluster in this case is called a *partitioned cluster*. The network in this situation is also called a *sundered network*.

RSCT

All subsystems of RSCT will remain active on each partition.

Within each partition, Topology Services will recreate its heartbeat rings, excluding adapters that are not reachable, and keep monitoring all network adapters in the cluster. The reliable messaging library will be updated to indicate the loss of connectivity; adapters that do not belong to this partition are assumed not to be alive. Group Services will be informed about the loss of connectivity to

the nodes of the other partition and ‘dissolve’ membership of them in the GPFS group. For each group, it will keep providing its services to the active members in a partition. There is no synchronization between the instances of the same group that are run on distinct partitions.

After network connectivity has been reestablished, the subsystems of RSCT will join their operating domains and continue to function without visible interruption. Topology Services will reconnect the disjoint heartbeat rings to form rings that include the corresponding adapters of all partitions. Group Services will merge instances of each group into one group. This is actually done by dissolving all but one instance of each group and letting the clients on nodes, for which an instance has been dissolved, rejoin the corresponding group.

Quorum in GPFS nodesets

If a GPFS nodeset is affected by a partition, this leads to the unmounting of the file system in all but one partition (or all, if none of them maintains quorum).

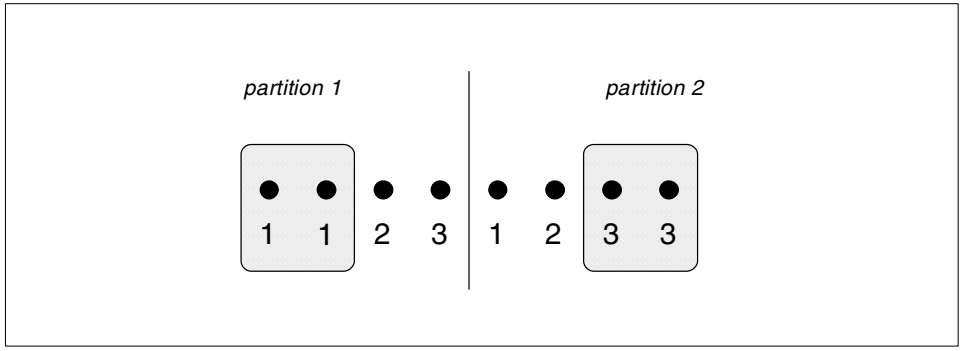


Figure 3-11 Partitioned cluster and quorum in GPFS nodesets

Figure 3-11 shows a GPFS cluster of eight nodes that is partitioned into two sets of four nodes. Three GPFS nodesets exists, with nodeset IDs 1 through 3. The nodes of the first nodeset belong to partition 1 maintaining quorum with the nodes of the third nodeset in partition 2.

The second nodeset contains two nodes. At first, both nodes will maintain quorum, if quorum is configured. If the file system is mounted on one node, this node will attempt to fence the other node out from the disks, which will cause the GPFS daemon on the other node to leave the active state. If the GPFS daemons on both nodes are active and the file system is mounted, the surviving node is the one which first succeeds with the recovery actions.

HACMP/ES

The HACMP/ES Cluster Manager will continue to operate on all partitions. After the network connectivity has been reestablished, the nodes on all partitions, except the one with the highest priority, will halt. A **`halt -q`** is performed as part of the script clexit.rc, which is run. This may have a drastic impact on the performance of GPFS and may lead to a loss of quorum for a GPFS nodeset leading to the unmounting of file systems on all nodes. This should be a very rare scenario if redundant networking connections are configured.

Disk fencing and quorum in a partitioned cluster

Figure 3-12 illustrates how disk fencing is used in conjunction with quorum to maintain data integrity in a partitioned cluster. A network failure has lead to a partitioned cluster, the partitions consist of nodes A,B,C, and D,E.

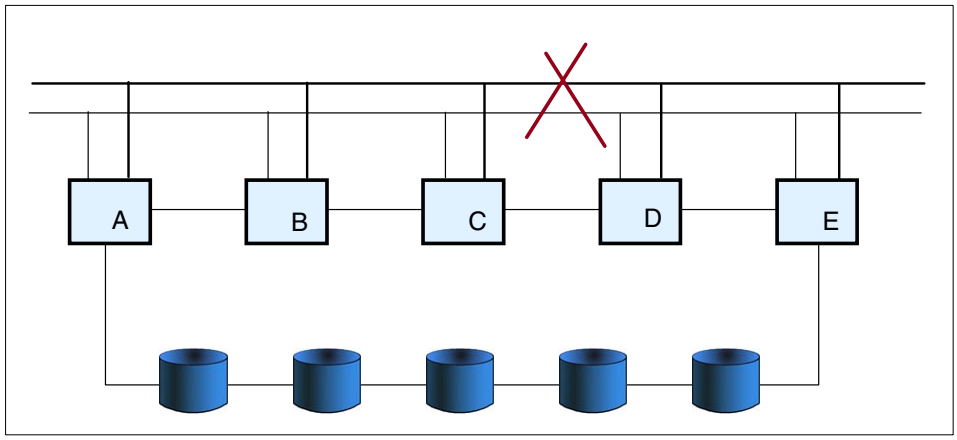


Figure 3-12 A partitioned cluster as a result of network failure

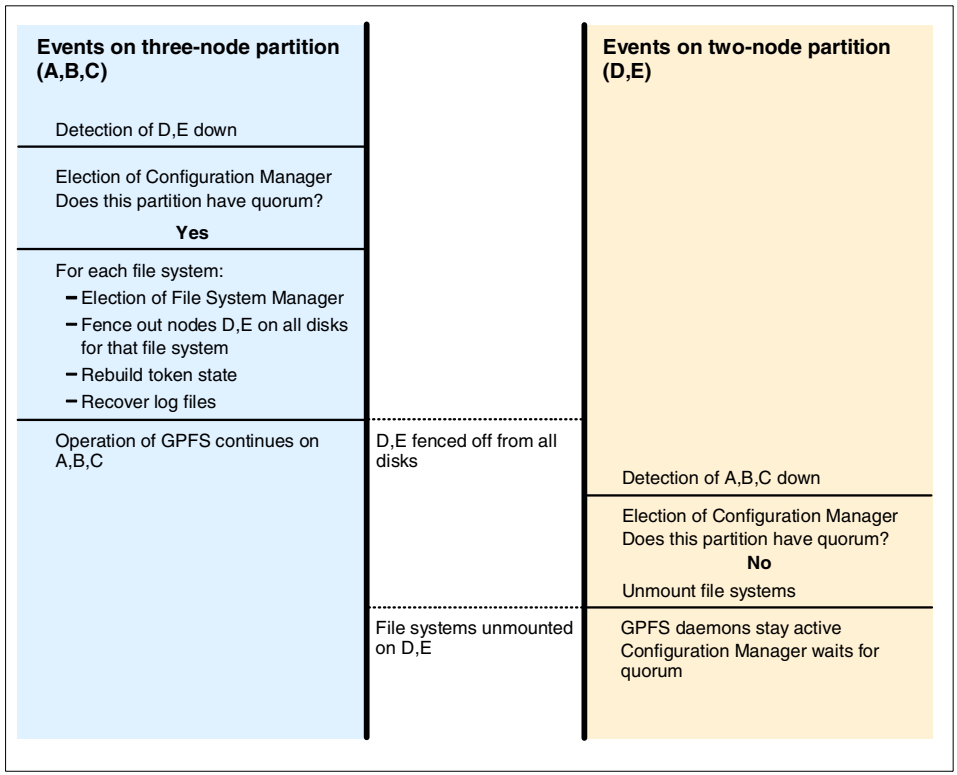


Figure 3-13 Events on partitioned clusters

Figure 3-13 shows two sequences of recovery actions that are run without synchronization after the cluster had become partitioned. The sequence of events starts when GPFS runs a protocol to perform recovery actions for both nodes, D and E.

First it is determined if a new configuration manager needs to be elected. The configuration manager elects a new file system manager, if needed; a Group Services barrier is reached. Afterwards, the file system manager nodes start the recovery of all file systems, which involves fencing out nodes D and E, rebuilding the token state, and updating the log files. The loss of connectivity to the nodes of the corresponding other partition will not be detected at the same time by both partitions.

At this point, a new configuration manager is elected, if necessary. The configuration manager establishes that the nodes in this partition do not have quorum and unmounts the file systems. When the recovery actions in both partitions are not synchronized, disk fencing in conjunction with quorum helps to maintain data integrity in a partitioned cluster. The nodes in the second partition cannot access the disks after the recovery actions have been completed on the first partition.

3.6 HACMP/ES overview

HACMP/ES is a clustering solution for AIX that provides high availability for application programs. HACMP stands for High Availability Cluster Multi-Processing. ES for Enhanced Scalability. The latter refers to the newer version of HACMP that supports a cluster size up to 32 nodes in an SP environment and 16 in a cluster of RS/6000 machines.

An application is highly available if it runs in a monitored system environment and, in case of a failure, is restored within a short time span. High availability is cheaper to implement than fault tolerance and satisfies the requirements of applications for which a downtime in the range of a few minutes is acceptable.

High availability in an HACMP/ES cluster is realized by:

- A redundant hardware configuration: To provide high availability, a cluster needs to contain two or more nodes and redundant networking connections between them. The nodes serve each other as standby. If a node fails, other nodes take over the applications than were active on that node.
- The distributed systems, that are part of HACMP/ES, that monitor the cluster and initiate recovery actions when a failure has been detected. They are:
  - Topology Services
  - Group Services
  - Event Management
  - HACMP/ES Cluster Manager
  - HACMP/ES SMUX Peer Daemon
  - HACMP/ES Cluster Information Services
  - HACMP/ES Cluster Lock Manager

The HACMP/ES Cluster Manager is the subsystem that drives the recovery actions to provide high availability for resources. The HACMP/ES SMUX Peer Daemon and the HACMP/ES Cluster Information Services are to provide other applications of utilities of HACMP/ES with information about the cluster. The HACMP/ES Cluster Lock Manager provides a locking protocol for applications that concurrently access shared external data.

Typically an application requires supporting system resources, such as a TCP/IP configuration for the access of clients, volume groups, etc. System resources can be associated with an application. On the cluster nodes, on which the application is active, those system resources will be configured.

An HACMP/ES cluster can be configured to provide high availability for multiple applications, depending on the number of nodes in the cluster and the requirements of each application onto the system resources.

3.6.1 Configuring HACMP/ES

Conceptually, the HACMP/ES configuration data are subdivided into two categories, the *cluster topology* and *cluster resources*. This configuration data is stored in HACMP specific object classes in the Object Data Manager (ODM).

Cluster topology

The cluster topology has already been introduced in Section 3.4.1, “Configuration of the cluster topology” on page 46.

A cluster adapter is of one of the following three types:

boot

A boot adapter is the primary adapter on a node for a given cluster network. It can be replaced by a service IP label belonging to the same network.

service

A service adapter is either a primary adapter on a node for a given network or a network interface configuration that will replace the configuration of a boot of standby adapter on the network to which it belongs. In the latter case, the service adapter is referred to as service IP label and is part of a resource group. If a service adapter is the primary adapter on a node for a given network, it cannot be moved to another node or replaced by another service label.

standby

Any further adapter, besides a boot or services adapter, that exists on a given network is a standby adapter. A service IP label can replace a standby adapter.

Cluster resources

The *cluster resources* are the entity of system resources that are made highly available by HACMP/ES.

System resources that are made highly available in HACMP/ES belong to a *resource group*. All resources belonging to a resource group are configured on a node on which the resource group is online. Each resource group has a nodeset defined, which contains the nodes on which the resource group can be brought online. A resource group can be brought online on one or multiple nodes, depending on the node relationship attribute for the nodeset. Further attributes of the nodeset determine the choice of the node within the nodeset, on which the resource group is brought online.

Example 3-6 shows the resource group definition of HACMP/ES. The resource group in this example, RES1, has two nodes, A and B. The service IP label, svc_1, will be configured on a cluster adapter of the node where the resource group is online. Furthermore, the volume group vg_1 will be varied on on that node, and the application APP1 will be started.

Example 3-6 SMIT screen for the configuration of a resource group

```
A:/> smitty hacmp
   Cluster Configuration
      Cluster Resources
         Change/Show Attributes of a Resouce Group

[TOP]                                                [Entry Fields]
  Resource Group Name                               RES1
  Node Relationship                                 cascading
  Participating Node Names                          A B

Service IP Label(s)                                [svc_1] +
Filesystems                                         []                          +
Filesystems Consistency Check                        fsck +
Filesystems Recovery Method                          sequential +
Filesystems/Directories to Export                   []                          +
Filesystems/Directories to NFS Mount                []                          +
Network For NFS Mount                               []                          +
Volume Groups                                       [vg_1] +
Concurrent Volume Groups                            []                          +
Raw Disk PVIDs                                      []                          +
AIX Connections Services                            []                          +
AIX Fast Connect Services                           []                          +
Application Servers                                 [APP1] +
Highly Available Communication Links                []                          +
Miscellaneous Data                                  []

Inactive Takeover Activated                          false                      +
Cascading Without Fallback Enabled                   false                      +
9333 Disk Fencing Activated                          false                      +
SSA Disk Fencing Activated                          false                      +
```

```
Filesystems mounted before IP configured            false
+
```

More than one instance of each resource type can be configured in a resource group. For example, a resource group could contain multiple volume groups.

Multiple resource groups can be configured in a cluster, depending on the number of nodes in the cluster and the resources that are configured in each group.

Configuration Limitations when GPFS is present

An HACMP/ES cluster that is used for GPFS can be further configured to make other applications highly available. There exists one restriction onto the cluster configuration: Any adapters that are used for the TCP/IP socket of the GPFS daemon cannot participate in an IP Address Takeover.

The resource group in the above example has a service IP label configured. If the resource group goes online on a node, the service IP label will replace a boot or standby adapter of the same network. This is referred to as IP Address Takeover.

Cluster verification and synchronization

Cluster verification and synchronization validate a cluster configuration and distribute it to all cluster nodes.

Cluster verification

To function as a distributed system and to provide recovery for the cluster resources, the HACMP/ES Cluster Manager depends on the following:

- The cluster manager daemons on all nodes have the same cluster configuration data, which is referred to as the cluster being synchronized.
- The cluster configuration corresponds to the real, existing system resources. For instance, adapters belonging to resource groups, which are configured in the cluster topology or volume groups, are configured correctly on AIX.

Cluster verification checks the above conditions. A cluster that failed cluster verification is not guaranteed to function reliably. In a cluster that is not synchronized, an attempt of a cluster manager daemon to join the active cluster manager subsystem will fail, if its cluster configuration is different from the one known to the active cluster. Some configuration errors may not be detected immediately, but are usually detected during the runtime of the cluster. For example, you may not notice that a resource is not properly configured until a node attempts to acquire a resource group and the acquisition of that particular resource is not possible.

Cluster synchronization
Cluster synchronization entails the distribution of a cluster configuration that is locally present on a node to all nodes in the cluster. During cluster synchronization the HACMP ODMs of the node from which the command is issued, are copied to the other nodes in the cluster. Furthermore, the ODMs are supplied with values that are specific to the configuration of system resources on each node.

After any change to the cluster configuration or the system resources that relate to cluster configuration data, the cluster configuration needs to be synchronized. Verification is run by default during cluster synchronization. The cluster configuration can be synchronized while the cluster manager is active on some nodes.

3.6.2 Error Recovery

When a failure is detected, an event is enqueued. The event scripts that are run for any cluster event perform the recovery actions.

Currently, in version 4.4 of HACMP/ES, error recovery is implemented for failures of the following

cluster nodes
If a cluster node fails, all resource groups that are online on that node are moved to another node. This is performed by the event scripts associated with the node_down event.

cluster adapter
If a service or boot adapter fails, the IP address of that adapter is reconfigured on the standby adapter by the event scripts associated with the swap_adapter_event.

applications
If an application that is subscribed to application monitoring fails, the resource group containing this application is moved to another node, by the events scripts associated with the rg_move event.

Partitioned Cluster

In a partitioned cluster, the nodes in each partition will detect the other side as down. The cluster will remain active within each partition; quorum is not implemented. Node down events are run for the nodes in the corresponding partition. If multiple partitions contain nodes of a resource group, that resource group will be online on multiple nodes as a result of the partition.

If the network connections in a partitioned cluster are reestablished and both partitions can again communicate with each other, the nodes belonging to the partition that has lower priority will halt.

Customization of error recovery

HACMP/ES provides recovery for failures in the cluster topology. Using application monitoring, a cluster configuration can be customized to provide recovery for other resources that are part of a resource group. It is a trend in the development of high availability clustering products to extend the range of system resources for which recovery from failures is provided. In versions of HACMP/ES newer than version 4.4, error recovery may be extended to cover system resources that are not discussed here.

4

Planning for implementation

In this chapter we explain why we made certain design choices for our GPFS cluster and then elaborate on exactly what those choices were. Each topic is split into two parts. The first part of a topic covers a general range of decisions that had to be made and the input to those decisions. The second part of each topic details what we decided so later chapters can configure the cluster in that manner. The one absolute we had to abide by was to use GPFS 1.4 without PSSP, making our implementation a non-SP environment. The resulting configuration is called a GPFS cluster.

4.1 Software

We divided our software discussion into two sections. The first section is a general discussion on the software options while the second section covers our specific software installation.

4.1.1 Software options

Software encompasses the AIX operating system, PSSP, HACMP/ES and GPFS 1.4. This section does not include any configuration of our application software, just the results of our installation choices.

SP environment

GPFS 1.4 can operate within an SP environment with or without VSDs as long as PSSP is version 3 release 2 or later on the control workstation. For the operating system, AIX is required to be at version 4 release 3.3 (with APAR IY12051) or later on the control workstation. PSSP provides group services to GPFS 1.4, without which it will not function in an SP environment.

Non-SP environment

For GPFS 1.4 in a non-SP environment, the necessary services are provided by RSCT (reliable scalable cluster technology), which is packaged with HACMP/ES Version 4.4.0 (5765-E54) with PTF IY12984 or later. The AIX operating system should again be at version 4 release 3.3 (5765-C34) with APAR IY12051 or later.

4.1.2 Software as implemented

The following sections describes which LPPs (licensed program products) we installed on our four RS/6000 machines and how we generated those displays.

Operating system

AIX 4.3.3 with APAR IY12051 or later was required for our non-SP implementation of GPFS clustering. We knew we were at AIX 4.3.3 by running **`oslevel`**. The important item to learn was: Did we have the required APAR? This APAR cross-referenced to PTF U473336 which is in the AIX 4.3.3 maintenance level 05. The cross-reference information is available from:

`http://techsupport.services.ibm.com/rs6k/fixdb.html`

We confirmed this level by running **instfix -ik 433-05_AIX_ML**, determined any missing filesets by running **instfix -ciqk 4330-05_AIX_ML | grep ":-:"** and corrected the inappropriate filesets. We could not use the simple command **instfix -i -k "IY12051"** which will report "not installed" (even if it is installed) since this is a packaging APAR number only and disappears as soon as it is installed.

HACMP/ES

The required level for HACMP/ES is version 4.4.0 (5765-E54) with PTF IY12984 or later modifications. To determine if the PTF specified is installed, run the **instfix** command, as shown in Example 4-1, and make sure all filesets are found.

Example 4-1 Verification of required code level of HACMP/ES

```
host1t:/> instfix -i -k "IY12984"
    All filesets for IY12984 were found.
```

We had to install HACMP/ES from scratch on all four of our nodes. One method is to acquire the code and run **smitty install** separately on each node. What we chose to do was to create a NFS mounted file system we called /tools/images and made it available to all nodes from host1t. We ran **bffcreate** to produce install images of the desired software and simultaneously installed it across the nodes. For a detailed explanation see Appendix B, "Distributed software installation" on page 207.

GPFS 1.4

We installed GPFS 1.4 in the same manner that we installed HACMP/ES, by using an NFS mounted filesystem mounted on every node with an image of the desired application software created there by running **bffcreate**. Verification was as easy as running **lslpp**, as shown in Example 4-2, since it had to be at level 1.4. Prerequisites only applied to AIX and HACMP/ES since that was the software that GPFS was specifically dependent on.

Example 4-2 Verification of GPFS installation

```
host1t:/> lslpp -l | grep mmfs
  mmfs.base.cmds             3.3.0.0  COMMITTED  GPFS File Manager Commands
  mmfs.base.rte              3.3.0.0  COMMITTED  GPFS File Manager
  mmfs.gpfs.rte              1.4.0.0  COMMITTED  GPFS File Manager
  mmfs.base.rte              3.3.0.0  COMMITTED  GPFS File Manager
  mmfs.gpfs.rte              1.4.0.0  COMMITTED  GPFS File Manager
  mmfs.gpfsdocs.data         3.3.0.0  COMMITTED  GPFS Server Manpages and
```

4.2 Hardware

We divided our discussion on the hardware requirements into two sections, first the options available to a new installation and second, is how we implemented the hardware into our environment.

4.2.1 Hardware options

While GPFS 1.4 operates in the traditional SP environment using an SP switch, this implementation exercised the operating environment outside the SP on a set of networked RS/6000 machines.

This implementation used two networks: A100 Mb/sec Ethernet and a 16 Mb/sec Token Ring. There are no VSDs outside of the SP environment, so this cluster used SSA disks directly attached to each node.

Host systems

The minimum hardware for GPFS 1.4 is a processor that is able to run AIX 4.3.3, enough spare disk space for the additional filesets required by the application software, an SSA adapter and at least one 100Mb/sec network adapter.

SSA disks and cabling diagram

There are many types of SSA adapters and disks, each with their own device driver and microcode. Its important to keep drivers and microcode current; see `http://www.hursley.ibm.com/ssa/index.html` for detailed information on code for your SSA adapters and disks.

There is a limit of eight SSA adapters to a single SSA loop. If a configuration requires more disk space than is supported by a single SSA adapter then multiple SSA adapters can be used in each node, with each SSA adapter, attached to separate loops. This will allow GPFS configurations in the cluster environment to support a considerable amount of disks dependent on the number of SSA adapters the server node can support. Some adapters have smaller limits; refer to *Understanding SSA Subsystems in Your Environment,* SG24-5750 for more information on this topic.

If SSA hardware RAID arrays are used, this limits the number of adapters in a loop to two. This is called an SSA hardware limit as shown in Table 4-1 on page 65. The use of the SSA RAID function is therefore limited to two-node clusters.

A more flexible approach to increased data availability in an eight-node cluster would be to cable the SSA disks across two separate SSA adapters in each node. The disks on each adapter would form a GPFS failure group allowing the configuration of GPFS data and metadata replication.

Table 4-1 SSA 4-P adapter table

Device	Minimum	Maximum
SSA disks	0 per loop	48 per loop, 96 per adapter
SSA adapters - Node	1 per node	Node H/W dependent
SSA adapters - Loop	1 per loop	8 per loop
Nodes	1 per cluster	8 per cluster
RAID arrays	1 per cluster	2 per cluster

4.2.2 Hardware

This section is about how we connected the nodes.

Host systems

The four systems were identical except that host1t and host2t had three internal SCSI disks while host3t and host4t had only two internal SCSI disks. This was not a factor in performance or set up, just noteworthy since hdisk addressing is not identical between the nodes. We used the spare SCSI disk in host1t as a shared resource between all four nodes mounted over NFS and called it /tools.

- 4 RS/6000 F50s with the following hardware:
  - 4 x CPUs each
  - 1.5GB memory each
  - 1 x 10/100 ethernet adapter 9-P
  - 1 x token ring adapter 9-O
  - 1 x SSA Adapter 4-P

SSA disks and cabling diagram

The SSA adapters were cabled into a 7133-D40 SSA disk drawer. We chose to use both the A and B loops available on the SSA adapter cards. SSA bandwidth is 40 MB/sec per loop in each direction for a total of 80MB/sec per loop and 160 MB/sec per adapter. The D40 SSA drawer was completely filled with sixteen 9.1GB SSA disk drives so they were split evenly between the two loops. We could view the speed of the two loops by running the ssa_speed command. This command could also be effective when troubleshooting a slow SSA loop.

```
host1t:/> ssa_speed -l pdisk0
40 40
```

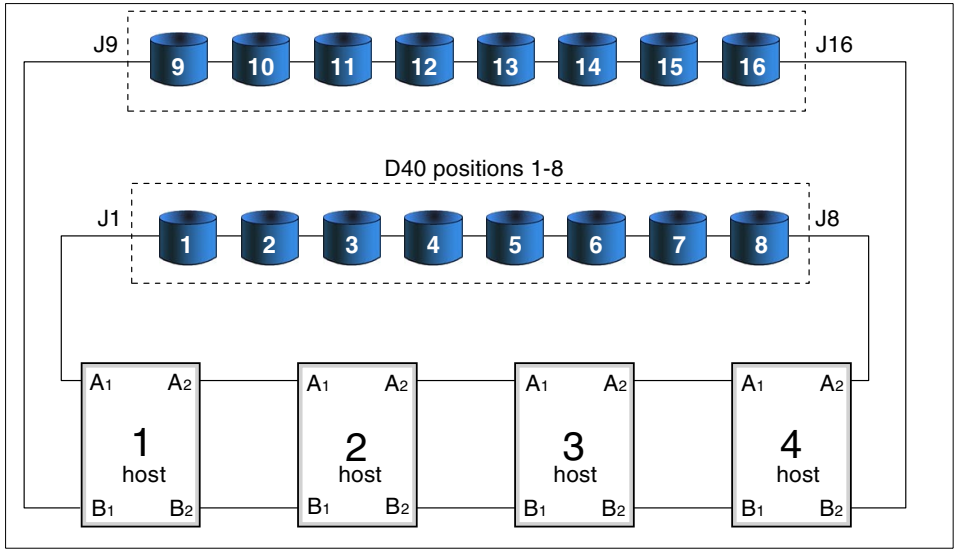


Figure 4-1 SSA Cabling

4.3 Networking

GPFS requires an IP network over which the GPFS socket connections will be established. This network should be dedicated to GPFS only.

4.3.1 Network options

For GPFS 1.4 to function, we had to have a dedicated IP network of at least 100Mb/sec bandwidth, faster being better. On this network the socket connections for the GPFS daemons will be established. The network will be configured to belong to the HACMP/ES cluster topology as well, in order to monitor the adapters for failures. This network is used by GPFS daemons for communicating file system state data. The actual file system data travels over the SSA adapters directly between nodes and disks.

The second IP network is used for user application traffic and will not be used by GPFS state data traffic.

The network defined for use by GPFS must not be a network designated for IP address takeover by HACMP. It must be an adapter with only a service address and no associated boot address. This network is assigned to GPFS with the **mmcrcluster** command.

4.3.2 Network

The token ring network is the default network and our access to the outside world. The second network is a 100 Mb ethernet that we dedicated to GPFS. To simplify the network, we named the nodes host1 thru host4 and tacked on a letter e for ethernet and a letter t for token ring addresses, as shown in Example 4-3. The local ethernet adapters are connected to an 8 port Alteon 180 hub while the token ring was attached to the laboratory network. A .rhosts file is required on all nodes. Figure 4-2 on page 68 is a schematic diagram of our network configuration.

Example 4-3 Listing of the adapter addresses for our nodes

```
host1t:/> cat /etc/hosts
127.0.0.1               loopback localhost      # loopback (lo0) name/address
9.12.0.21       host1t host1t.itso.ibm.com
9.12.0.22       host2t host2t.itso.ibm.com
9.12.0.23       host3t host3t.itso.ibm.com
9.12.0.24       host4t host4t.itso.ibm.com
129.40.12.129   host1e host1e.itso.ibm.com
129.40.12.130   host2e host2e.itso.ibm.com
129.40.12.131   host3e host3e.itso.ibm.com
129.40.12.132   host4e host4e.itso.ibm.com

host1t:/> cat /.rhosts
host1t
host2t
host3t
host4t
host1e
host2e
host3e
host4e
```

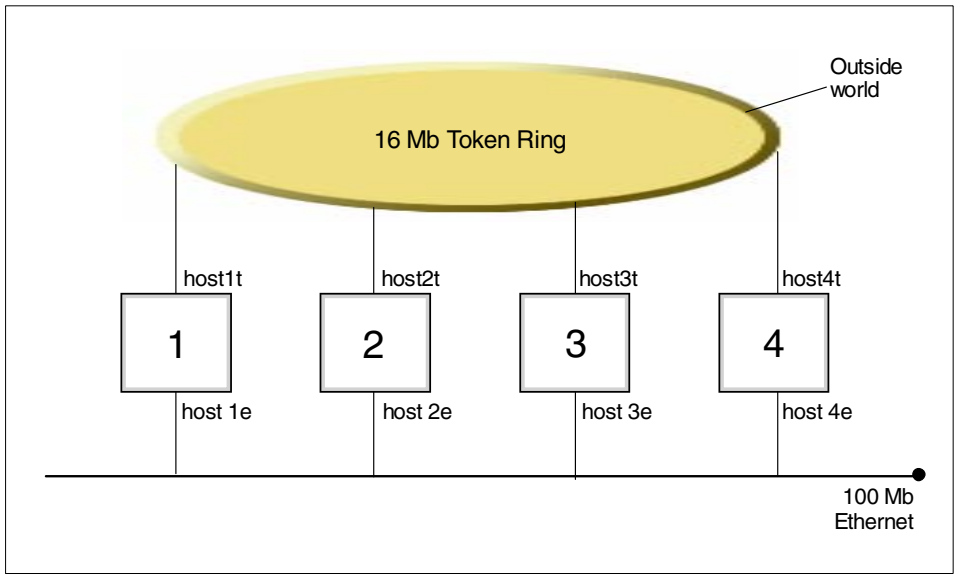


Figure 4-2 Network configuration

4.4 High availability

The remainder of this chapter contains high availability considerations that apply to the configuration of networks for HACMP/ES and SSA disk arrays.

4.4.1 Networks

Networking requirements to support GPFS

HACMP/ES requires redundant network connections between nodes. Otherwise a single network failure potentially could cause a system halt on a subset of nodes in the cluster, as explained in Section 3.5.3, “Partitioned clusters” on page 51.

Our network configuration contains two IP networks. Often, in the hardware setup of an HACMP/ES cluster, a serial network such as RS232 or Target Mode SSA is used as the second redundant network connection between hosts. In our case, to support GPFS, this is not recommended. Serial networks are slow and error recovery performed over serial networks takes more time than over an IP network. In most uses of HACMP/ES ‘this does not affect things. However, in GPFS, fast processing of failures is important.

Requirements for networks used in HACMP/ES

The following are important considerations regarding the network configuration used in an HACMP/ES cluster. They are not specific to the presence of GPFS.

Single points of failure in the networking hardware

To eliminate single points of failure, the HACMP/ES cluster should have redundant network connections. It should be verified that the networking hardware does not contain hidden single points of failures. Networking connections are not redundant if they share hardware components, such as networking switches, adapters, or cables.

Do not send heartbeats into the wide world

Networks used for HACMP/ES should provide a local connection between hosts. A networking connection that requires keepalive signals to be sent over a campus network, for instance, does not satisfy the requirements of HACMP/ES.

Requirements for the HACMP/ES GPFS dedicated network

As already discussed, the network dedicated to GPFS should be used exclusively by GPFS. If the GPFS network is used for other purposes as well, this could have an adverse effect on the overall GPFS performance. If HACMP/ES is used to provide high availability for other applications, the following limitation exists:

The adapters that support the TCP/IP socket connection for GPFS must not be configured for IP Address Takeover in HACMP/ES.

All nodes within a GPFS nodeset have access to the same GPFS file systems without the need for IP Address Takeover of the GPFS network adapter.

Any application that fails over to another HACMP node in the same GPFS nodeset continues to have access to the same GPFS file systems as the failed node.

In the HACMP/ES cluster topology an adapter has one of three functions, *boot*, *standby*, or *service,* that determine the role of an adapter in relation to IP Address Takeover.

A network that is used in an HACMP/ES cluster typically has two or more adapter connections with each host, whereby one is configured as *boot* and all further as *standby adapters*.

For our purposes, a single adapter is sufficient on each network, since HACMP/ES is not used to provide high availability of the TCP/IP socket connection established by GPFS itself. If only one network adapter that connects a host to a network is present, then this adapter, due to the HACMP/ES

configuration rules, will be configured in the HACMP/ES cluster topology as a *service* adapter. The network adapter over which the TCP/IP socket connection for GPFS is established, could be configured as boot or a standby adapter. However, if it is configured as service adapter, we are sure that it cannot be used 'by accident' for IP Address Takeover, since such a configuration would not pass cluster verification. See *HACMP 4.3: Enhanced Scalability Installation and Administration Guide,* Vol.1, SC23-4284-02 for more details regarding the configuration of networks adapters.

4.4.2 SSA configuration

There are two possibilities for disk data replication, using RAID groups or configuring *failure groups*.

A failure group in GPFS is a group of disks that share a common point of failure. All disks that belong to an SSA adapter are in the same failure group. The disks in SSA loops that do not share host adapters constitute different failure groups. Hence, to use failure groups for GPFS in a cluster, at least two SSA host adapters are required on each node. When configuring the file system, disks can be assigned to failure groups. Then, replication of user data and/or metadata can be specified for that file system.

For more details refer to *GPFS for AIX: Concepts, Planning, and Installation Guide*.

Our configuration does not plan for data replication due to the fact that the implementation only has one SSA adapter per node.

5

Configuring HACMP/ES

This chapter details the steps that are necessary to configure a HACMP/ES cluster for GPFS and shows it by example for our configuration. A cluster configuration for GPFS only requires the definition of a cluster topology.

In this chapter, we will discuss:

- Requirement onto the system environment
- Configuring the HACMP/ES cluster topology
- Starting and stopping the HACMP/ES cluster services
- Monitoring the cluster

5.1 Prerequisites

Sections 5.1.1, 5.1.2, and 5.1.3 describe the prerequisites that we need before starting to configure the HACMP/ES cluster.

5.1.1 Security

Before starting the configuration of HACMP/ES, we need to verify the remote access permissions regarding the system security.

Remote access permissions

On each node, the file /.rhosts needs to contain the names of all service and boot adapters in the cluster. The operation of the cluster services daemons itself does not depend on remote access permissions, however utilities and administrative commands require that remote access is allowed to all cluster nodes as shown in Example 5-1. We verify that on all cluster nodes the /.rhosts file contains the following aliases:

Example 5-1 Aliases for all nodes in /.rhosts file

```
host1t/> more /.rhosts
host1t
host2t
host3t
host4t
host1e
host2e
host3e
host4e
```

5.1.2 Network configuration

We verify that the network configuration corresponds to our network as planned in “Network” on page 67, and that name resolution is configured correctly.

Network interfaces

The correct configuration of network interfaces is crucial for HACMP/ES. The following need to be verified:

- All network interfaces that are used to configure cluster adapters are in the up state, as ascertained by the **netstat** command.
- The configuration of network interfaces corresponds to the one present at boot time.
- All network interfaces that will be configured as cluster adapters belong to the same IP subnet.

We verify that the interface configuration contains all host adapters. The **netstat** command is issued on all hosts.

Example 5-2 Verify network adapters are up

```
host1t:/> netstat -i
Name  Mtu   Network     Address              Ipkts Ierrs    Opkts Oerrs  Coll
en1   1500  link#2      0.4.ac.3e.b3.95     857150     0   864472   289     0
en1   1500  129.40.12.1 host1e              857150     0   864472   289     0
tr0   1492  link#3      0.4.ac.ad.ce.50    1412205     0  1287386     0     0
tr0   1492  9.12        host1t             1412205     0  1287386     0     0
lo0   16896 link#1                          325031     0   325563     0     0
lo0   16896 127         loopback           325031     0   325563     0     0
lo0   16896 ::1                            325031     0   325563     0     0
```

The network interface configuration, as shown above, needs to correspond to the configuration present at boot time.

In Example 5-3, we confirm that all adapters of one network type belong the same IP subnet.

Example 5-3 Verify boot time configuration

```
host1t:/> for i in 1 2 3 4; do
> rsh host"$i"t netstat -i | grep host"$i"e
> done
en1   1500  129.40.12.1 host1e              185615     0   177040  1211     0
en1   1500  129.40.12.1 host2e              749881     0   717379  3057     0
en1   1500  129.40.12.1 host3e              765077     0   707196     0     0
en1   1500  129.40.12.1 host4e              858264     0   895507  5860     0

host1t:/> for i in 1 2 3 4; do
> rsh host"$i"t netstat -i | grep host"$i"t
> done
tr0   1492  9.12        host1t             1296792     0  1434347     0     0
tr0   1492  9.12        host2t             1721115     0  1348656     0     0
tr0   1492  9.12        host3t              177266     0   148358     0     0
tr0   1492  9.12        host4t              324299     0   250125     0     0
```

Name resolution

Aliases need to be configured for all IP addresses that will be used as cluster adapters.

It is good practice to keep the /etc/hosts files on all cluster nodes identical and to use a naming convention for cluster adapters that reflects the adapter function and network membership. Example 5-4 lists all of the aliases we have defined for our cluster adapters.

Example 5-4 Verify aliases for all cluster adapters

```
host1t:/> more /etc/hosts

127.0.0.1        loopback localhost       # loopback (lo0) name/address
9.12.0.21       host1t host1t.itso.ibm.com
9.12.0.22       host2t host2t.itso.ibm.com
9.12.0.23       host3t host3t.itso.ibm.com
9.12.0.24       host4t host4t.itso.ibm.com
129.40.12.129   host1e host1e.itso.ibm.com
129.40.12.130   host2e host2e.itso.ibm.com
129.40.12.131   host3e host3e.itso.ibm.com
129.40.12.132   host4e host4e.itso.ibm.com
```

Name resolution should be configured such that name lookup is first attempted locally. The name lookup sequence is determined by the hosts keyword in the /etc/netsvcs.conf file as shown in Example 5-5.

Example 5-5 netsvcs.conf file

```
host1t:/> more /etc/netsvcs.conf
hosts = local , bind
```

5.1.3 System resources

HACMP/ES subsystems and GPFS generate log files that are stored in /var. The size of /var should be sufficiently large; the RSCT subsystems will not start unless there is enough free space available in /var. In Example 5-6, we verify that our /var file has enough free space to start the RSCT subsystem.

Example 5-6 Checking size of /var

```
host1t:/> df -s /var
Filesystem    1024-blocks     Free* %Used    Iused %Iused Mounted on
/dev/hd9var         32768     15492   53%      524     7% /var
```

5.2 Configuring the cluster topology

When configuring the cluster topology, all configurations are first performed on one node. Afterwards, cluster synchronization is necessary to distribute the configuration to all other nodes.

The following steps in the configuration of the cluster topology have to be performed in the order below:

1. Cluster Name and ID
2. Cluster Nodes
3. Cluster Adapters

5.2.1 Cluster Name and ID

When more than one HACMP/ES cluster is created on a set of hosts that share a common network, the cluster names and IDs within that node set must be unique. In Example 5-7, we use the **smitty hacmp** command to define both the cluster name and ID number.

Example 5-7 Defining the cluster name and ID

```
host1t:/> smitty hacmp
   Cluster Configuration
      Cluster Topology
         Configure Cluster
            Add a Cluster Definition
            (we choose the following values)

Cluster ID            111
Cluster Name          hcluster
```

The cluster definition can be modified at any point the cluster is not active on any node.

5.2.2 Cluster nodes

After having defined a cluster name, the set of nodes that will belong to the cluster can be specified. The node set can be modified at any later point.

One or more node names can be entered, separated by whitespaces as shown in Example 5-8.

Example 5-8 Defining cluster node names

```
host1t:/> smitty hacmp
   Cluster Configuration
      Cluster Topology
         Configure Nodes
            Add Cluster Nodes
            (the screen will include the following entries)

Node Names         host1t host2t host3t host4t
```

5.2.3 Cluster adapters

By configuring an adapter for HACMP/ES, it is assigned to the network topology, hence is monitored for failures and used for the communication in the cluster. Each adapter will be specified to belong to a cluster network, indicating the connectivity between adapters. We choose the following names for our cluster networks:

gpfs_net For the adapters belonging to the Ethernet
noname_net For the adapters belonging to the Token Ring network

We configure the adapters below in Example 5-9.

Example 5-9 Defining cluster adapters

```
host1t:/> smitty hacmp
   Cluster Configuration
      Cluster Topology
         Configure Adapter
            Add an Adapter
            (the screen will include the following entries)

Adapter IP label          host1e
Network Type              ether
Network Name              gpfs_net
Network Attribute         public
Adapter Function          service
Node Name                 host1t
```

The ‘entries’ above mean the following:

Adapter IP Label: The alias of the IP address of the adapter, as it appears in /etc/hosts, or the IP address of the adapter.

Network Type: Refers to the physical type of the network, such as Ethernet or Token Ring. PF4 generated a list of supported type.

Network Name: The user defined name of the network to which this adapter belongs.

Network Attribute: Refers to the function that a cluster network can have. The network attribute can have the value public, private, or serial. We defined our networks as public.

Adapter Function: A cluster adapter can be a *boot*, *service*, or *standby* adapter. We chose service as adapter function, see “Requirements for the HACMP/ES GPFS dedicated network” on page 69.

Node Name: The node to which this adapter belongs. The node must belong to the set of cluster nodes that we configured above.

Example 5-10 shows the configuration of an adapter for the network noname_net.

Example 5-10 Configuration of cluster adapter

```
Adapter IP label          host1t
Network Type              token
Network Name              noname_net
Network Attribute         public
Adapter Function          service
Node Name                 host1t
```

In Example 5-11, we add the remaining three adapters from the command line.

Example 5-11 Adding adapters via the command line

```
host1:/> for i in 2 3 4; do
> /usr/es/sbin/cluster/utilities/claddnode -a host"$i"e:ether\
> :gpfs_net:public:service::-n host1t
> done
host1:/> for i in 2 3 4; do
> /usr/es/sbin/cluster/utilities/claddnode -a host"$i"t:token\
> :noname_net:public:service::-n host1t
> done
```

5.2.4 Displaying the cluster topology

After the configuration has been performed locally on host1t, we verify that our configuration is correct with the **smitty hacmp** command as shown in Example 5-12.

Example 5-12 Display cluster topology

```
host1t:/> smitty hacmp
   Cluster Configuration
      Cluster Topology
         Show Cluster Topology
            Show Topology Information by Network Name
               Show all Networks
               (the screen will look as follows)

gpfs_net        public     host1t     host1e
                           host2t     host2e
                           host3t     host3e
                           host4t     host4e

noname_net      public     host1t     host1t
                           host2t     host2t
                           host3t     host3t
                           host4t     host4t
```

5.3 Verification and synchronization

In the last section, the cluster topology was defined on the local node, host1t. The configuration now needs to be synchronized as shown in Example 5-13, in order for all other nodes in the cluster to recognize it. Cluster synchronization needs to be performed again after any change to the cluster configuration and also after changing certain system resources. Cluster synchronization, by default, entails that cluster verification is run first, which validates that our configuration does not contain conflicting settings that would cause the cluster services to exit abnormally. If cluster verification detects an error, the **smitty hacmp** command will not succeed and will give details about the configuration errors that were found.

Example 5-13 .Synchronize cluster topology

```
host1t:/> smitty hacmp
   Cluster Configuration
      Cluster Topology
         Synchronize Cluster Topology
         (the screen will include the following entries)

Ignore Cluster Verification Errors?          No
Emulate or Actual?                           actual
Skip Cluster Verification                    No
```

Note: The cluster topology always needs to be synchronized.

5.3.1 Cluster resource configuration

A configuration of HACMP/ES for GPFS does not entail defining cluster resources. It is possible to use HACMP/ES for other applications besides GPFS. See “Requirements for the HACMP/ES GPFS dedicated network” on page 69 for details.

5.4 Starting the cluster

Before starting the cluster, we made sure that the system times are synchronized on all nodes. In Example 5-14, we use the gdsh utility to simultaneouly set the clocks on all the nodes. The subsystem of RSCT relies on a time synchronization between all cluster nodes.

Example 5-14 Synchronize clocks on all cluster nodes

```
host1t:/> gdsh setclock host1t
host1t: Thu Feb 22 10:14:27 2001
host2t: Thu Feb 22 10:14:28 2001
```

```
host3t: Thu Feb 22 10:14:28 2001
host4t: Thu Feb 22 10:14:28 2001
```

The gdsh utility is covered in Appendix C, “A useful tool for distributed commands” on page 211.

Starting the cluster services on a node entails the startup of the HACMP/ES cluster manager daemon and all subsystems on which it depends.

In Example 5-15, we start the cluster services on node host1t.

Example 5-15 Start cluster services

```
host1t:/> smitty hacmp
   Cluster Services
      Start Cluster Services
      (the screen will include the following entries)

Start now, on system restart or both         now
BROADCAST message at startup?                true
Startup Cluster Lock Services?               false
Startup Cluster Information Daemon?          true
```

The previous smit screen corresponds to the following command:

```
host1t:/> /usr/es/sbin/cluster/etc/rc.cluster -boot -N -b -i
```

The above command starts the daemons of the RSCT subsystems, the HACMP/ES cluster manager, clstrmgrES, and the clinfoES daemon.

A successful completion of **rc.cluster** only indicates that all daemons have been started. We still have to wait for all events in context with the startup of the cluster node to finish successfully.

On node host1t, we issue the **clstat** command, as shown in Example 5-16, to monitor the state of the cluster on all nodes.

Example 5-16 Monitor cluster status

```
host1t:/> /usr/es/sbin/cluster/clstat

               clstat - HACMP for AIX Cluster Status Monitor
               ---------------------------------------------

Cluster: hcluster       (111)           Thu Feb 22 10:26:01 EST 2001
               State: UP               Nodes: 4
               SubState: STABLE
```

```
Node: host1t            State: UP
   Interface: host1e (0)              Address: 129.40.12.129
                                      State:   UP
   Interface: host1t (1)              Address: 9.12.0.21
                                      State:   UP

Node: host2t            State: DOWN
   Interface: host2e (0)              Address: 129.40.12.130
                                      State:   DOWN
   Interface: host2t (1)              Address: 9.12.0.22
                                      State:   DOWN

Node: host3t            State: DOWN
   Interface: host3e (0)              Address: 129.40.12.131

                                      State:   DOWN
   Interface: host3t (1)              Address: 9.12.0.23
                                      State:   DOWN

Node: host4t            State: DOWN
   Interface: host4e (0)              Address: 129.40.12.132
                                      State:   DOWN
   Interface: host4t (1)              Address: 9.12.0.24
                                      State:   DOWN


********************** f/forward, b/back, r/refresh, q/quit *****************
```

After awhile, the output of the above command should indicate that the cluster is stable, which indicates that no more events are enqueued to be processed.

Alternatively, we can convince ourselves that the start of the cluster services has been successful by inspecting the log files as in Example 5-17.

Example 5-17 Check cluster log files

```
host1t:/> more /usr/es/sbin/cluster/history/cluster.02222001
Feb 22 10:24:56 EVENT START: node_up host1t
Feb 22 10:24:58 EVENT COMPLETED: node_up host1t
Feb 22 10:24:58 EVENT START: node_up_complete host1t
Feb 22 10:24:59 EVENT COMPLETED: node_up_complete host1t
```

The output of the above command shows that the `node_up` and `node_up_complete` event have completed. These are the two events that are processed by all members when a node joins the HACMP/ES cluster.
In Example 5-18 on page 81 we can now start the cluster services on the remaining three cluster nodes simultaneously.

Example 5-18 Simultaneously start cluster services on multiple nodes

```
host1t:/> gdsh -w host2t,host3t,host4t \
> '/usr/es/sbin/cluster/etc/rc.cluster -boot -N -b -i'
```

The Group Services subsystem will serialize the requests of the daemons to join the clstrmgrES subsystem. After all daemons have joined, the cluster will return to the stable state.

A look at the log file, as shown in Example 5-19, illustrates the serialization of events by Group Services. When the cluster daemon on a node joins the clstrmgrES subsystem, a node_up and a node_up_complete event are run.

The file on all nodes will show the same order of events.

Example 5-19 Check cluster log files

```
host1t:/> more /usr/es/sbin/cluster/history/cluster.02222001
Feb 22 10:24:56 EVENT START: node_up host1t
Feb 22 10:24:58 EVENT COMPLETED: node_up host1t
Feb 22 10:24:58 EVENT START: node_up_complete host1t
Feb 22 10:24:59 EVENT COMPLETED: node_up_complete host1t
Feb 22 10:40:42 EVENT START: node_up host2t
Feb 22 10:40:42 EVENT COMPLETED: node_up host2t
Feb 22 10:40:45 EVENT START: node_up_complete host2t
Feb 22 10:40:45 EVENT COMPLETED: node_up_complete host2t
Feb 22 10:41:13 EVENT START: node_up host3t
Feb 22 10:41:14 EVENT COMPLETED: node_up host3t
Feb 22 10:41:16 EVENT START: node_up_complete host3t
Feb 22 10:41:16 EVENT COMPLETED: node_up_complete host3t
Feb 22 10:41:43 EVENT START: node_up host4t
Feb 22 10:41:44 EVENT COMPLETED: node_up host4t
Feb 22 10:41:47 EVENT START: node_up_complete host4t
Feb 22 10:41:48 EVENT COMPLETED: node_up_complete host4t
```

The subsystems of RSCT, Topology Services, Group Services, and Event Management are started when starting the cluster manger subsystem. Once the node_up event for a node has been started, we know that Topology Services and Group Services have become active.

5.5 Monitoring the cluster

The state of the cluster and the event history usually give enough information to verify that the cluster is operational.

5.5.1 The clstat command

The **clstat** command, as shown in Example 5-20, gives information about the state of the cluster as a distributed subsystem and the state of individual nodes and adapters.

Example 5-20 Cluster monitoring

```
host1t:/> /usr/es/sbin/cluster/clstat


               clstat - HACMP for AIX Cluster Status Monitor
               ---------------------------------------------

Cluster: hcluster       (111)           Thu Feb 22 17:11:21 EST 2001
               State: UP               Nodes: 4
               SubState: STABLE

        Node: host1t            State: UP
           Interface: host1e (0)                Address: 129.40.12.129
                                                State:   UP
           Interface: host1t (1)                Address: 9.12.0.21
                                                State:   UP

        Node: host2t            State: DOWN
           Interface: host2e (0)                Address: 129.40.12.130
                                                State:   DOWN
           Interface: host2t (1)                Address: 9.12.0.22
                                                State:   DOWN

        Node: host3t            State: UP
           Interface: host3e (0)                Address: 129.40.12.131
                                                   State:   UP
           Interface: host3t (1)                Address: 9.12.0.23
                                                State:   UP

        Node: host4t            State: DOWN
           Interface: host4e (0)                Address: 129.40.12.132
                                                State:   DOWN
           Interface: host4t (1)                Address: 9.12.0.24
                                                State:   DOWN

********************** f/forward, b/back, r/refresh, q/quit *****************
```

Cluster state

The cluster state refers to the state of the cluster manager as a distributed subsystem on all nodes. It can have the values *up*, *down*, and *unknown*.

Cluster substate

The cluster substates listed below give detailed information about the cluster.

stable

- The cluster services are active on at least one node.
- The cluster manager subsystem is in an error free state.
- No events are currently processed.

unstable

- The cluster services are active on at least one node.
- The cluster manager is currently processing events.

unknown

- The state is not determined.

reconfig

- The cluster services are active on one or more nodes in the cluster. On one node an event is running and has exceeded the maximum time limit that is assumed for an event to be run. This is likely due to a failure while executing an event script.
- No new event can be processed; user intervention is necessary.

error

- The cluster services are active on at least one node.
- An error has occurred and no events can be processed.

Node state

The state of a node can have one of the following four values.

up	The node is part of the cluster
down	The node is not part of the cluster
joining	The node is joining the cluster
leaving	The node is leaving the cluster

Adapter state

The state of an adapter can have the following values:

up	The adapter is currently configured, and detected as alive by Topology Services
down	The adapter is currently not configured, or a failure of it has been detected.

The **`clstat`** command requires that the clinfoES subsystem is active on the node on which it is issued.

5.5.2 Event history

The file /usr/es/sbin/cluster/history/cluster.mmddyyyy gives a list of all events that have been run on a day, specified by the date in the filename suffix. Events are the units for Group Services barrier synchronization in the cluster. Therefore, the files on all nodes will give the same information regarding the sequence of events.

```
host1t:/> more /usr/es/sbin/cluster/history/cluster.02222001
```

5.5.3 Monitoring HACMP/ES event scripts

HACMP/ES event scripts are Korn shell scripts that are run on the cluster nodes to perform the configuration changes that are required by a certain event. The file /tmp/hacmp.out contains the output of the event scripts that are run on the local node. See Example 5-21 for a sample output of /tmp/hacmp.out on host4t; note that the example omits a part of the output.

Example 5-21 Execution of event scripts

```
host4t:/> more /tmp/hacmp.out

Feb 22 10:41:43 EVENT START: node_up host4t

node_up[124] [[ high = high ]]
node_up[124] version=1.10.1.23
node_up[125] node_up[125] cl_get_path
HA_DIR=es
node_up[127] NODENAME=host4t
node_up[129] HPS_CMD=/usr/es/sbin/cluster/events/utils/cl_HPS_init
node_up[130] VSD_CMD=/usr/lpp/csd/bin/hacmp_vsd_up1
node_up[131] SS_FILE=/usr/es/sbin/cluster/server.status
   .
   .
node_up[170] [ -n  -a -f /usr/es/sbin/cluster/events/utils/cl_HPS_init ]
node_up[228] [ REAL = EMUL ]
node_up[233] cl_ssa_fence up host4t
cl_ssa_fence[70] [[ high = high ]]
cl_ssa_fence[70] version=1.9
cl_ssa_fence[71] cl_ssa_fence[71] cl_get_path
HA_DIR=es
cl_ssa_fence[74] echo PRE_EVENT_MEMBERSHIP=host1t host2t host3t
PRE_EVENT_MEMBERSHIP=host1t host2t host3t
cl_ssa_fence[75] echo POST_EVENT_MEMBERSHIP=host1t host2t host3t host4t
POST_EVENT_MEMBERSHIP=host1t host2t host3t host4t
cl_ssa_fence[77] EVENT=up
cl_ssa_fence[78] NODENAME=host4t
cl_ssa_fence[79] STATUS=0
```

```
cl_ssa_fence[82] export EVENT_ON_NODE=host4t
cl_ssa_fence[84] [ 2 -gt 1 ]
cl_ssa_fence[91] [ host4t = host4t ]
cl_ssa_fence[93] [ host1t host2t host3t !=  ]
cl_ssa_fence[95] exit 0
node_up[236] [ 0 -ne 0 ]
node_up[316] rm -f /tmp/.NFSSTOPPED
node_up[317] rm -f /tmp/.RPCLOCKDSTOPPED
node_up[356] [ host4t != host4t ]
node_up[375] exit 0
Feb 22 10:41:44 EVENT COMPLETED: node_up host4t


Feb 23 10:41:47 EVENT START: node_up_complete host4t

node_up_complete[62] [[ high = high ]]
node_up_complete[62] version=1.1.3.18
node_up_complete[63] node_up_complete[63] cl_get_path
HA_DIR=es
node_up_complete[65] NODENAME=host4t
   .
   .
node_up_complete[263] exit 0
Feb 22 10:41:48 EVENT COMPLETED: node_up_complete host4t
```

The commands that are executed during an event depend on the cluster configuration and the system environment.

The file /tmp/hacmp.out is the main diagnostic source for errors that occur during an event. If an event fails, one can easily localize the command that caused the failure of the event.

5.5.4 Monitoring the subsystems

All subsystems of HACMP/ES are subscribed to the system resource controller. We can monitor them using the **lssrc** command as shown in Example 5-22.

Example 5-22 List HACMP/ES subsystems

```
host1t:/> lssrc -a | grep active

(the output will include the following)
topsvcs          topsvcs          20460   active
grpsvcs          grpsvcs          20538   active
grpglsm          grpsvcs          21218   active
emsvcs           emsvcs           21432   active
```

```
emaixos          emsvcs           20850   active
clstrmgrES       cluster          17938   active
clsmuxpdES       cluster          22204   active
clinfoES         cluster          19640   active
```

For a complete listing of all subsystems belonging to HACMP/ES, see Appendix E, “Subsystems and Log files” on page 229.

For the subsystems belonging to the RSCT component, a long listing is generated by the **lssrc** command as shown in Example 5-23. We look at the output **lssrc** generated for the Group Services subsystem:

Example 5-23 Group services subsystem

```
host1t:/> lssrc -ls grpsvcs
Subsystem         Group            PID     Status
 grpsvcs          grpsvcs          19162   active
3 locally-connected clients.  Their PIDs:
21218(hagsglsmd) 21432(haemd) 17938(clstrmgr)
HA Group Services domain information:
Domain established by node 1
Number of groups known locally: 3
                  Number of   Number of local
Group name        providers   providers/subscribers
ha_em_peers             4           1           0
CLRESMGRD_111           4           1           0
CLSTRMGR_111            4           1           0
```

The above contains the groups in which clients of Group Services on host1t participate. Three clients exist on host1t, hagsglsmd, haemd, and clstrmgr.

5.5.5 Log files

The log files that are generated by the daemons of the cluster group and HACMP/ES utilities can be found in the below smit menu in Example 5-24.

Example 5-24 Show a cluster log directory

```
host1t:/> smitty hacmp
   Cluster System Management
      Cluster Log Management
         Change/Show a Cluster Log Directory
```

The most commonly used log files are:

- /tmp/hacmp.out

  Generated by HACMP/ES event scripts

- /usr/es/adm/cluster.log

  Generated by cluster scripts and daemons

- /usr/es/sbin/cluster/history/cluster.mmdd

  Cluster history file generated daily

Further log files exist, that contain debugging information. For a complete list of all log files maintained by all subsystems of RSCT and HACMP/ES, see Appendix E, “Subsystems and Log files” on page 229.

5.6 Stopping the cluster services

Stopping the cluster services entails terminating all subsystems in a controlled way and reversing changes to the AIX configuration that were made when starting the cluster. Example 5-25 shows the use of the **smitty hacmp** command to stop the cluster services.

Example 5-25 Stop cluster services–**smitty**

```
host1t:/> smitty hacmp
   Cluster Services
      Stop Cluster Services
      (the screen will include the following entries)

Stop now, on system restart or both       now
BROADCAST cluster shutdown?               true
Shutdown mode                             graceful
```

Shutdown mode specifies what reconfigurations of the system environment will be performed when resource groups are online on this node. We do not have resource groups configured, hence the setting for this value in our case will not make a difference.

The above smit screen in Example 5-25 corresponds to the following **clstop** command in Example 5-26.

Example 5-26 Stop cluster services–**clstop**

```
host1t:/> /usr/es/sbin/cluster/utilities/clstop -y -N -g
Feb 22 2001 18:22:41/usr/es/sbin/cluster/utilities/clstop: called with flags -y
-N -g

Broadcast message from root@host1t (tty) at 18:22:41 ...

HACMP/ES for AIX on  shutting down.
Please exit any cluster applications...
```

```
0513-044 The clstrmgrES Subsystem was requested to stop.
0513-044 The clinfoES Subsystem was requested to stop.
0513-044 The clsmuxpdES Subsystem was requested to stop.
```

A successful completion of the above command only indicates that a request to stop has been issued successfully to all daemons. We still have to wait until all events in context with the stopping of the cluster node have successfully finished. In Example 5-27, we use the lssrc command to verify that all of the cluster subsystems are inoperative.

Example 5-27 Check status of cluster subsystems

```
host1t:/> lssrc -g cluster
Subsystem         Group            PID     Status
 clstrmgrES       cluster                  inoperative
 clsmuxpdES       cluster                  inoperative
 clinfoES         cluster                  inoperative
```

Note that after successful exit of the cluster manager, the subsystems of RSCT will terminate as well as shown in Example 5-28.

Example 5-28 Check status of RSCT subsystems

```
host1t:/> lssrc -a | grep svcs
 topsvcs          topsvcs                  inoperative
 grpsvcs          grpsvcs                  inoperative
 grpglsm          grpsvcs                  inoperative
 emsvcs           emsvcs                   inoperative
 emaixos          emsvcs                   inoperative
```

In Example 5-29, we stop the cluster services on the remaining three nodes using the Cluster Single Point of Control (CSPOC) facility.

Example 5-29 Stop cluster services

```
host1t:\> smitty hacmp
   Cluster system Management
      HACMP for AIX Cluster Services
         Stop Cluster Services
         (the screen will include the following entries)

* Stop now, on system restart or both           now
  Stop Cluster Services on these nodes          [host2t,host3t,host4t]
  BROADCAST cluster shutdown?                   true
* Shutdown mode                                 graceful
      (graceful or graceful with takeover, forced)
```

We verify that the cluster services have stopped on all nodes, by issuing the above command **lssrc -g cluster** on the remaining three nodes, which will show that the subsystems of the cluster group have become inactive.

When a node leaves the HACMP/ES cluster, a node_down, and node_down complete event are run on all active cluster nodes. The sluster log file in Example 5-30 shows these events starting and completing.

Example 5-30 Check cluster log files

```
host2t:/> more /usr/es/sbin/cluster/history/cluster.02222001
...
Feb 22 10:41:43 EVENT START: node_up host4t
Feb 22 10:41:44 EVENT COMPLETED: node_up host4t
Feb 22 10:41:47 EVENT START: node_up_complete host4t
Feb 22 10:41:48 EVENT COMPLETED: node_up_complete host4t
Feb 22 17:50:13 EVENT START: node_down host1t graceful
Feb 22 17:50:14 EVENT COMPLETED: node_down host1t graceful
Feb 22 17:50:15 EVENT START: node_down_complete host1t graceful
Feb 22 17:50:17 EVENT COMPLETED: node_down_complete host1t graceful
Feb 22 17:50:28 EVENT START: node_down host2t graceful
Feb 22 17:50:28 EVENT COMPLETED: node_down host2t graceful
Feb 22 17:50:29 EVENT START: node_down_complete host2t graceful
Feb 22 17:50:31 EVENT COMPLETED: node_down_complete host2t graceful
```

6

Configuring GPFS and SSA disks

This chapter steps you through the necessary commands to create volume groups, logical volumes, and the various mmfs configurations that allowed us to finally mount a GPFS file system. Our choice of technique and scripts does not reflect the only way to perform the following tasks; it is meant to be simple to follow.

- Create the GPFS cluster
- Create the nodeset
- Start GPFS
- Create the SSA volume groups and logical volumes
- Create and mount the GPFS file system

6.1 Create the GPFS cluster

Make sure that all appropriate HACMP subsystems are running by either observing that the /usr/es/sbin/cluster/clstat indicators are all green or running **lssrc -a | egrep "clstrmgrES|grpsvcs|topsvcs"** on every node, as shown in Example 6-1. Do not proceed unless the subsystems are active, refer to Chapter 5, "Configuring HACMP/ES" on page 71. Use of the **lssrc -ls grpsvcs** as explained in Section 5.5.4, "Monitoring the subsystems" on page 85 is also useful.

Example 6-1 Verify HACMP subsystems are running

```
host1t:/> lssrc -a | egrep "clstrmgrES|grpsvcs|topsvcs"
 topsvcs          topsvcs          20460   active
 grpsvcs          grpsvcs          20538   active
 grpglsm          grpsvcs          21218   active
 clstrmgrES       cluster          17938   active
```

6.1.1 Create the GPFS nodefile

In our case we created a file /tools/gpfs_config/gpfs_nodefile that was visible to every node where we expected to mount this file system. Again we used our NFS mounted /tools file system to simplify this task. An alternative would be to create the gpfs_nodefile (an arbitrary name) on every node; the contents of this file is listed in Example 6-2. The names selected in this file reflect the ethernet addresses of these nodes in our dedicated network for GPFS internal communications. We designated a secondary sever and a primary server node, but we could have also created a file that limited the nodes that could be File System Manager nodes. That file is called /var/mmfs/etc/cluster.preferences and would contain a simple list of eligible nodes.

Example 6-2 Ethernet node names

```
host1t:/tools/gpfs_config> cat gpfs_nodefile
host1e
host2e
host3e
host4e
```

6.1.2 Create the cluster commands

Example 6-3 on page 93 illustrates how to use the **mmcrcluster** command to create a GPFS cluster from a set of nodes.

- **mmcrcluster**
  - **-t hacmp** type of cluster

– **-p host1e** primary GPFS configuration data server name (pick any)

– **-s host2e** secondary GPFS configuration data server name (pick any)

– **-n /tools/gpfs_config/gpfs_node** our nodefile location and name (list of node descriptors)

Example 6-3 Create GPFS cluster

```
host1t:/> mmcrcluster -t hacmp -p host1e -s host2e -n  \
/tools/gpfs_config/gpfs_node

mmcrcluster: Command successfully completed
mmcrcluster: Propagating the changes to all affected nodes.
This is an asynchronous process.
```

To verify that this command was successful, run the **mmlscluster** command, as shown in Example 6-4.

Example 6-4 List GPFS cluster information

```
host1t:/> mmlscluster

GPFS cluster information
========================

GPFS system data repository servers:
------------------------------------
  Primary server:    host1e
  Secondary server:  host2e

Cluster nodes that are not assigned to a nodeset:
-------------------------------------------------
  1  host1e  129.40.12.129  host1e
  2  host2e  129.40.12.130  host2e
  3  host3e  129.40.12.131  host3e
  4  host4e  129.40.12.132  host4e
```

If you are unhappy with the GPFS cluster you created, run **mmdelcluster**, as in Example 6-5, to remove the cluster and start over.

Example 6-5 Delete GPFS cluster

```
host1t:/tools/gpfs_config> mmdelcluster -n gpfs_nodefile
mmdelcluster: Command successfully completed
```

6.2 Create the nodeset

Within the GPFS cluster we just defined we need to include one or more GPFS nodesets. Any node can only be in one nodeset at a time but can be transferred between nodesets when desired.

6.2.1 Create dataStrucureDump

We decided a better place for the mmfs dumps was under the /tmp file system. This must be created on all nodes. In Example 6-6, we used **gdsh "mkdir /tmp/mmfs"** to make the directory and we inform the configuration about it in the **mmconfig** command.

Example 6-6 Create mmfs dumps directory

```
host1t:/> gdsh "mkdir /tmp/mmfs"
host1t:/>

host1t:/> gdsh "ls -l /tmp/mmfs"
host1t: total 0
host2t: total 0
host3t: total 0
host4t: total 0
```

6.2.2 The mmconfig command

This commands defines a new GPFS nodeset and configures GPFS prior to creating the file. We defined the following parameters:

- **mmconfig**
  - **-n gpfs_nodefile** (NodeFile)
  - **-A** autostart GFS daemons is yes
  - **-C set1 (**NodesetId**,** any name will do)
  - **-D /tmp/mmfs** (dataStructureDump**,** dump file location)
  - **-p 80M** (pagepool size)

After running the **mmconfig** command, we ran **mmlsnode -a** and **mmlsconfig**, as shown in Example 6-7 on page 95, to verify proper configuration. Notice that Filesystems in nodeset set1 replies with (none). We also checked for the existence of /etc/cluster.nodes, /var/mmfs/gen/mmsdrfs and /var/mmfs/etc/mmfs.cfg, three very important files for GPFS.

Example 6-7 Verify configuration

```
host1t:/tools/gpfs_config> mmconfig -n gpfs_nodefile -A -C set1 -D  \
/tmp/mmfs -p 80M

mmconfig: Command successfully completed
mmconfig: Propagating the changes to all affected nodes.
This is an asynchronous process.
host1t:/tools/gpfs_config> mmlsnode -a
GPFS nodeset    Node list
------------  -------------------------------------------------
   set1         host1e host2e host3e host4e
host1t:/tools/gpfs_config> mmlsconfig
Configuration data for nodeset set1:
------------------------------------
pagepool 80M
dataStructureDump /tmp/mmfs
multinode yes
autoload yes
useSingleNodeQuorum no
wait4RVSD no
comm_protocol TCP
clusterType hacmp
group Gpfs.set1
recgroup GpfsRec.set1
File systems in nodeset set1:
-----------------------------
(none)
```

6.3 Start GPFS

Start the subsystem for GPFS with the **mmstartup** command, as shown in Example 6-8, on one node (which will then propagate to all the nodes in the named set), or use the **startsrc -s mmfs** on all appropriate nodes. Next, verify the processes are running on all appropriate nodes by **gdsh "lssrc -s mmfs"**. We cannot create a file system at this point until we have defined the volume groups and logical volumes for the SSA drives.

Example 6-8 Start GPFS subsystem

```
host1t:/tools/gpfs_config> mmstartup -C set1
Fri Feb 9 11:21:32 EST 2001: mmstartup: Starting GPFS ...
host1e:  0513-059 The mmfs Subsystem has been started. Subsystem PID is 24176.
host2e:  0513-059 The mmfs Subsystem has been started. Subsystem PID is 17626.
host3e:  0513-059 The mmfs Subsystem has been started. Subsystem PID is 11910.
host4e:  0513-059 The mmfs Subsystem has been started. Subsystem PID is 14846.
```

```
host1t:/tools/gpfs_config> gdsh "lssrc -s mmfs"
host1t: Subsystem         Group            PID     Status
host1t:  mmfs             aixmm            24176   active
host2t: Subsystem         Group            PID     Status
host2t:  mmfs             aixmm            17626   active
host3t: Subsystem         Group            PID     Status
host3t:  mmfs             aixmm            11910   active
host4t: Subsystem         Group            PID     Status
host4t:  mmfs             aixmm            14846   active
```

6.4 Create the SSA volume groups and logical volumes

We had to perform several steps on all nodes in our cluster in order to make the SSA disks usable by GPFS.

A high level overview of these steps is:

- On one node, define volume groups and logical volumes on each disk for use by GPFS.
- On the remaining nodes in the GPFS cluster, import volume groups to make the logical volume definitions known to each node's ODM.

It is the logical volumes that are created and made available to each node that GPFS file systems are made up of. Only after the logical volumes are created and made available to each node, can GPFS file systems be created.

Since various machines can have different numbers of internal disks and SSA disks, we could not depend on the pdisk# and hdisk# being the same across all four of our nodes. We needed to know this to build volume groups so we could in turn build logical volumes that GPFS pointed to during its configuration step. The following is a brief outline of the steps needed to complete this task. Following this is a high level overview of the procedure for creating the volume groups and logical volumes.

- Perform on host1t only
  - **mkvg -n -f -s 16 -c -y gpfs(0-15) hdisk(x)** make volume group
  - **varyonvg gpfsvg(0-15)** vary on volume group
  - **mklv -b n -w n -y gpfslv(0-15) gpfsvg(0-15) 542** make logical volume
  - **varyoffvg gpfsvg(0-15)** vary off volume group

- Perform the following three steps completely on each node before moving on to the next node. This is only to be done on the rest of the nodes in the cluster (host2t, host3t and host4t in our case).
  - **importvg -y gpfs(0-15)** hdisk(x) import volume group
  - **chvg -a n gpfsvg(0-15)** make volume group non auto varyon
  - **varyoffvg gpfsvg(0-15)** vary off volume group

6.4.1 Create PVID list

The match between pdisks and hdisk is not always the same across different nodes. In Example 6-9, we queried each node to discover what node referred to pdisk0.

Example 6-9 Nodes and pdisk

```
host1t:/> gdsh “ssaxlate -l pdisk0”
host1t: hdisk3
host2t: hdisk3
host3t: hdisk2
host4t: hdisk2
```

Notice that pdisk0 is hdisk3 on two nodes (host1t and host2t) while it is hdisk2 on two other nodes (host3t and host4t). The only descriptor guaranteed to be consistent is the physical volume ID (PVID) which is the middle number in the **lspv** command, as shown in Example 6-10. Refer to Appendix A, “Mapping virtual disks to physical SSA disks” on page 201, for a detailed explanation of translating between pdisk numbers and hdisk numbers.

Example 6-10 List physical volumes

```
host1t:/> lspv
hdisk0          000b4a7df90e327d    rootvg
hdisk1          000b4a7de4b48b4f    rootvg
hdisk2          000b4a7d1075fdbf    toolsvg
hdisk3          000007024db58359    None
hdisk4          000007024db5472e    None
hdisk5          000007024db54fb4    None
hdisk6          000007024db5608a    None
hdisk7          000007024db571ba    None
hdisk8          000007024db5692c    None
hdisk9          000158511eb0f296    None
hdisk10         000007024db57a49    None
hdisk11         000007024db58bd3    None
hdisk12         000007024db53eac    None
hdisk13         000007024db5361d    None
hdisk14         000007024db51c4b    None
```

```
hdisk15         000007024db524ce    None
hdisk16         000007024db52d7b    None
hdisk17         000007024db513d2    None
hdisk18         000007024db55810    None
```

We can run **lspv | awk '{print $2}' > logvolpvid**, as shown in Example 6-11, to create a file with just PVIDs then eliminate the entries that are not the SSA disks we want for our file system (SCSI disks or other SSAs). This file can then be used to loop commands we need to define the volume groups properly. We also could have run **ssadisk -a ssa0 -L** and **ssadisk -a ssa0 -P** to make these determinations, but spare SSA disks would have to be removed manually.

Example 6-11 Create file with only PVIDs

```
host1t:/tools/ralph> lspv | awk '{print $2}' > logvolpvid

host1t:/tools/ralph> cat logvolpvid
000007024db58359
000007024db5472e
000007024db54fb4
000007024db5608a
000007024db571ba
000007024db5692c
000158511eb0f296
000007024db57a49
000007024db58bd3
000007024db53eac
000007024db5361d
000007024db51c4b
000007024db524ce
000007024db52d7b
000007024db513d2
000007024db55810
```

6.4.2 Make SSA volume groups

We made a volume group for every SSA disk attached to node host1t (16 disks), incrementing the volume group name and selecting the disks in the order they appeared in the **lspv** command and stored in the file logvolpvid.

- **mkvg**
  - **-n** do not vary on on restart
  - **-f** force
  - **-s 16** PP size
  - **-c** concurrent

– **-y gpfsvg(x)** volume group name, we'll vary the x from 0-15

When we ran the following script, shown in Example 6-12, we were able to create a volume group on all our disks.

Example 6-12 Make SSA volume groups

```
#!/usr/bin/ksh
j=0
for i in `cat logvolpvid`
do
hdisk=`lspv | grep $i | awk '{print $1}'`
mkvg -n -f -s 16 -c -y gpfs$j $hdisk
j=`expr $j + 1`

done
```

To verify the successful creation, run the **lspv** command, as shown in Example 6-13. Notice that the gpfsvg names start at 0 and continue to 15.

Example 6-13 Verify creation of SSA volume groups

```
host1t:/tools/paden/config> lspv
hdisk0          000b4a7df90e327d    rootvg
hdisk1          000b4a7de4b48b4f    rootvg
hdisk2          000b4a7d1075fdbf    toolsvg
hdisk3          000007024db58359    gpfsvg0
hdisk4          000007024db5472e    gpfsvg1
hdisk5          000007024db54fb4    gpfsvg2
hdisk6          000007024db5608a    gpfsvg3
hdisk7          000007024db571ba    gpfsvg4
hdisk8          000007024db5692c    gpfsvg5
hdisk9          000158511eb0f296    gpfsvg6
hdisk10         000007024db57a49    gpfsvg7
hdisk11         000007024db58bd3    gpfsvg8
hdisk12         000007024db53eac    gpfsvg9
hdisk13         000007024db5361d    gpfsvg10
hdisk14         000007024db51c4b    gpfsvg11
hdisk15         000007024db524ce    gpfsvg12
hdisk16         000007024db52d7b    gpfsvg13
hdisk17         000007024db513d2    gpfsvg14
hdisk18         000007024db55810    gpfsvg15
```

6.4.3 Vary on the volume groups

We now need to vary on the volume groups in order to build the logical volumes. Since the volume groups are now in numerical order (0 - 15), the simplest way to vary on all of them is with a while loop, as shown in Example 6-14 on page 100.

Example 6-14 Vary on the volume groups

```
#!/usr/bin/ksh
i=0
while [ "$i" -le 15 ]
do
varyonvg gpfsvg$i
i=`expr $i + 1`

done
```

To verify this operation was successful, we ran the **lsvg -o** command, as shown in Example 6-15, which only shows volume groups that are varied on, making sure all sixteen disks were present and ignoring other volume groups such as rootvg. We also ran the **lsvg gpfsvg(0-15)** command to view the details of each volume group. Pay particular attention to the line `Concurrent : Capable`, it's important all volume groups be concurrent capable or simultaneous sharing will not be possible.

Example 6-15 List volume groups

```
host1t:/tools/ralph> lsvg -o
gpfsvg15
gpfsvg14
gpfsvg13
gpfsvg12
gpfsvg11
gpfsvg10
gpfsvg9
gpfsvg8
gpfsvg7
gpfsvg6
gpfsvg5
gpfsvg4
gpfsvg3
gpfsvg2
gpfsvg1
gpfsvg0
toolsvg
rootvg

host1t:/tools/ralph> lsvg gpfsvg0
VOLUME GROUP:   gpfsvg0                  VG IDENTIFIER:  000b4a7d70d6b2e7
VG STATE:       active                   PP SIZE:        16 megabyte(s)
VG PERMISSION:  read/write               TOTAL PPs:      542 (8672 megabytes)
MAX LVs:        256                      FREE PPs:       0 (0 megabytes)
LVs:            1                        USED PPs:       542 (8672 megabytes)
OPEN LVs:       1                        QUORUM:         2
TOTAL PVs:      1                        VG DESCRIPTORS: 2
STALE PVs:      0                        STALE PPs:      0
```

```
ACTIVE PVs:     1                       AUTO ON:        no
Concurrent:     Capable                 Auto-Concurrent: Disabled
VG Mode:        Non-Concurrent
MAX PPs per PV: 1016                    MAX PVs:        32
```

6.4.4 Make logical volume

Example 6-16 shown the script we used to make GPFS logical volumes.

- **mklv**
  - **-b n** bad block relocation no
  - **-w n** mirror write no
  - **-y gpfslv(x)** select a logical volume name
  - **gpfsvg(x) 542** the maximum blocks a 9.1GB SSA disk can hold

Example 6-16 Make logical volume

```
#!/usr/bin/ksh
i=0
while [ "$i" -le 15 ]
do
mklv -b n -w n -y gpfslv$i gpfsvg$i 542
i=`expr $i + 1`

done
```

To verify success, we can run **lslv gpfslv(x)**, as shown in Example 6-17, against every GPFS logical volume we just created. Only gpfslv0 is displayed for simplicity.

Example 6-17 List GPFS logical volumes

```
host1t:/tools/ralph> lslv gpfslv0
LOGICAL VOLUME:     gpfslv0                VOLUME GROUP:   gpfsvg0
LV IDENTIFIER:      000b4a7d70d6b2e7.1     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       opened/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            542                    PP SIZE:        16 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                542                    PPs:            542
STALE PPs:          0                      BB POLICY:      non-relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: off
EACH LP COPY ON A SEPARATE PV ?: yes
```

6.4.5 Vary off the volume groups

When we finished defining the logical volumes, we had to vary off the volume groups from host1t. We used a while loop, as shown in Example 6-18, to address all the volume groups and vary them off and confirmed this was successful by running **lsvg -o**, as shown in Example 6-19, to make sure they were unavailable to host1t.

Example 6-18 Vary off the volume groups

```
#!/usr/bin/ksh
i=0
while [ "$i" -le 15 ]
do
varyoffvg gpfsvg$i
i=`expr $i + 1`

done
```

Example 6-19 List volume groups

```
host1t:/tools/ralph> lsvg -o
toolsvg
rootvg
```

6.4.6 Import the volume groups

- **importvg**
  - **-y gpfsvg(x)** name the volume group

This step, and the following four steps, must be done on each node completely before moving on to the next node. Again, we must match (synchronize) what the current node calls its PVID and its hdisk(x) with the gpfsvg(x) we gave that disk on the first node (host1t). An **lspv** shows that relationship but we want to assign the gpfsvg(0-15) IDs in the same order as the PVIDs appear in the file logvolpvid we created earlier. We will use the **gdsh** commands, so we must use absolute paths wherever required. Again, the NFS mounted /tools directory is very handy to make this information available to all nodes simultaneously. To import the volume groups onto the second node (host2t) run the following script, shown in Example 6-20 on page 103. Scripts can be named and placed in the /tools directory as we did, or they can be slightly modified and run directly from the command line.

Example 6-20 Import volume groups

```
#!/usr/bin/ksh
j=0
for i in `cat /tools/ralph/logvolpvid`
do
hdisk=`lspv | grep $i | awk '{print $1}'`
importvg -y gpfsvg$i $hdisk
j=`expr $j + 1`

done
```

We confirmed the success of this command by running **lsvg** on that node and observing that, along with the existing rootvg volume group, all sixteen GPFS volume groups should now exist. An alternate method is to run **lspv**, as shown in Example 6-21, which shows additional information. We do not need to run **varyonvg** at this point, since **importvg** automatically varies on the volume groups.

Example 6-21 List physical volumes

```
host1t:/tools/ralph> gdsh -w host2t "lspv"
host2t: hdisk0          000b4a9dcac1948a    rootvg
host2t: hdisk1          000b4a9dcac192ef    rootvg
host2t: hdisk2          000444527adfc8bd    None
host2t: hdisk3          000007024db58359    gpfsvg0
host2t: hdisk4          000007024db5472e    gpfsvg1
host2t: hdisk5          000007024db54fb4    gpfsvg2
host2t: hdisk6          000007024db5608a    gpfsvg3
host2t: hdisk7          000007024db571ba    gpfsvg4
host2t: hdisk8          000007024db5692c    gpfsvg5
host2t: hdisk9          000158511eb0f296    gpfsvg6
host2t: hdisk10         000007024db57a49    gpfsvg7
host2t: hdisk11         000007024db58bd3    gpfsvg8
host2t: hdisk12         000007024db53eac    gpfsvg9
host2t: hdisk13         000007024db5361d    gpfsvg10
host2t: hdisk14         000007024db51c4b    gpfsvg11
host2t: hdisk15         000007024db524ce    gpfsvg12
host2t: hdisk16         000007024db52d7b    gpfsvg13
host2t: hdisk17         000007024db513d2    gpfsvg14
host2t: hdisk18         000007024db55810    gpfsvg15
```

6.4.7 Change the volume group

- **chvg**
  - **-a n gpfs(x)** auto activation no

We cannot allow any node to grab the disks and mount them, so auto activation on all nodes for all GPFS disks must be set to no. That can be checked by running **lsvg gpfsvg(x) | grep AUTO** on every GPFS volume group and observing the word “no” follows AUTO ON. We already set that to no on host1t during the **mkvg** command, now we must also do it on the rest of the nodes in turn. In Example 6-22, another simple while loop will assist us. Run **gdsh -w host2t /tools/ralph/changevg,** the script changevg follows.

Example 6-22 Change volume groups

```
#!/usr/bin/ksh
i=0
while [ "$i" -le 15 ]
do
chvg -a n gpfsvg$i
i=`expr $i + 1`

done
```

6.4.8 Vary off the volume groups

In Example 6-23, the following while loop is identical to the one used earlier on host1t and the verification test is the same, run **lsvg -o** and make sure the volume groups are gone. We cannot export the volume group unless it is varied off.

Example 6-23 Vary off volume groups

```
#!/usr/bin/ksh
i=0
while [ "$i" -le 15 ]
do
varyoffvg gpfsvg$i
i=`expr $i + 1`

done
```

Step through all remaining nodes (in our case, do host3t then host4t, since we just did host2t).

6.5 Create and mount the GPFS file system

These final steps will bring everything else together.

6.5.1 Create a disk descriptor file

A descriptor must be passed to GPFS for every disk in the file system. This descriptor took the form of **DiskName::::FailureGroup.** In our case, shown in Example 6-24, the disk name was the logical volume name and since all disks are seen by all nodes, there can only be one failure group. This file must be available on all nodes. We elected to place this file in the NFS mounted /tools directory as /tools/gpfs_config/disk.desc and arbitrarily chose a failure group number of one.

Example 6-24 Disk descripton file

```
host1t:/tools/gpfs_config> cat disk.desc
gpfslv0::::1
gpfslv1::::1
gpfslv2::::1
gpfslv3::::1
gpfslv4::::1
gpfslv5::::1
gpfslv6::::1
gpfslv7::::1
gpfslv8::::1
gpfslv9::::1
gpfslv10::::1
gpfslv11::::1
gpfslv12::::1
gpfslv13::::1
gpfslv14::::1
gpfslv15::::1
```

6.5.2 Run the mmcrfs create file system command

In Example 6-25, the **mmcrfs** command will create the GPFS file system based on the attached parameters.

- **mmcrfs**
  - **/gpfs** mount point for the gpfs file system
  - **gpfs** the device name of the file system to be created
  - **-F disk.desc** points to file containing list of disk descriptors
  - **-C set1**the nodeset identifier you want the file system to belong to
  - **-A no** do not automatically mount the file system (personal choice)

Example 6-25 Create file system

```
host1t:/tools/gpfs_config> mmcrfs /gpfs1 gpfs1 -F disk.desc -C set1 -A no
The following disks of gpfs1 will be formatted on node host1t:
    gpfslv0: size 8880128 KB
```

```
    gpfslv1: size 8880128 KB
    gpfslv2: size 8880128 KB
    gpfslv3: size 8880128 KB
    gpfslv4: size 8880128 KB
    gpfslv5: size 8880128 KB
    gpfslv6: size 8880128 KB
    gpfslv7: size 8880128 KB
    gpfslv8: size 8880128 KB
    gpfslv9: size 8880128 KB
    gpfslv10: size 8880128 KB
    gpfslv11: size 8880128 KB
    gpfslv12: size 8880128 KB
    gpfslv13: size 8880128 KB
    gpfslv14: size 8880128 KB
    gpfslv15: size 8880128 KB
Formatting file system ...
Creating Inode File
Creating Allocation Maps
Clearing Inode Allocation Map
Clearing Block Allocation Map
  41 % complete on Fri Feb  9 11:53:20 2001
  86 % complete on Fri Feb  9 11:53:25 2001
 100 % complete on Fri Feb  9 11:53:26 2001
Flushing Allocation Maps
Completed creation of file system /dev/gpfs1.
All disks up and ready
mmcrfs: Propagating the changes to all affected nodes.
This is an asynchronous process.
```

In Example 6-26, we verified the file system was created by observing the new entry in the /etc/filesystems. We ran the following **cat** command on all nodes to make sure.

Example 6-26 Verify creation of the file system

```
host1t:/tools/gpfs_config> cat /etc/filesystems | grep -p /gpfs1
/gpfs1:
        dev             = /dev/gpfs1
        vfs             = mmfs
        nodename        = -
        mount           = false
        type            = mmfs
        account         = false
```

Another verification step is to run the **lslv -l gpfsls(x)** command on all nodes, as shown in Example 6-27. We built a while loop and ran it on every node. Only host1t is shown for simplification.

Example 6-27 List logical volumes

```
#!/usr/bin/ksh
i=0
while [ "$i" -le 15 ]
do
lslv -l gpfslv$i
i=`expr $i + 1`
done

host1t:/tools/ralph> testfs | pg
gpfslv0:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk3            542:000:000   19%           109:108:108:108:109
gpfslv1:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk4            542:000:000   19%           109:108:108:108:109
gpfslv2:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk5            542:000:000   19%           109:108:108:108:109
gpfslv3:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk6            542:000:000   19%           109:108:108:108:109
gpfslv4:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk7            542:000:000   19%           109:108:108:108:109
gpfslv5:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk8            542:000:000   19%           109:108:108:108:109
gpfslv6:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk9            542:000:000   19%           109:108:108:108:109
gpfslv7:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk10           542:000:000   19%           109:108:108:108:109
gpfslv8:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk11           542:000:000   19%           109:108:108:108:109
gpfslv9:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk12           542:000:000   19%           109:108:108:108:109
gpfslv10:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk13           542:000:000   19%           109:108:108:108:109
gpfslv11:N/A
PV                COPIES        IN BAND       DISTRIBUTION
```

```
hdisk14           542:000:000   19%           109:108:108:108:109
gpfslv12:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk15           542:000:000   19%           109:108:108:108:109
gpfslv13:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk16           542:000:000   19%           109:108:108:108:109
gpfslv14:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk17           542:000:000   19%           109:108:108:108:109
gpfslv15:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk18           542:000:000   19%           109:108:108:108:109
```

6.5.3 Mount the file system

Once the verifications are complete, run the **mount** command on all nodes by entering **gdsh "mount /gpfs1"**. Another mount command is **gdsh "mount -t mmfs"** which accomplishes the same thing. The simplest way to verify is a **df** command to all nodes and compare the results, as shown in Example 6-28. Finally **cd** to the new file system, **touch** a file then **rm** that file.

Example 6-28 Display file systems on all nodes

```
host1t:/tools/ralph> gdsh "df -k /gpfs1"
host1t: Filesystem    1024-blocks      Free %Used    Iused %Iused Mounted on
host1t: /dev/gpfs1      142077952 141965568    1%       12     1% /gpfs1
host2t: Filesystem    1024-blocks      Free %Used    Iused %Iused Mounted on
host2t: /dev/gpfs1      142077952 141965568    1%       12     1% /gpfs1
host3t: Filesystem    1024-blocks      Free %Used    Iused %Iused Mounted on
host3t: /dev/gpfs1      142077952 141965568    1%       12     1% /gpfs1
host4t: Filesystem    1024-blocks      Free %Used    Iused %Iused Mounted on
host4t: /dev/gpfs1      142077952 141965568    1%       12     1% /gpfs1

host1t:/tools/ralph> cd /gpfs1
host1t:/gpfs1> ls -l
total 0
host1t:/gpfs1> touch test
host1t:/gpfs1> ls -l
total 0
-rw-r--r--   1 root     system         0 Feb 13 16:38 test
host1t:/gpfs1> rm test
host1t:/gpfs1> ls -l
total 0
```

7

Typical administrative tasks

To keep a GPFS file system viable to its users requires monitoring, tuning, and alterations. The original mission for the file system will change over time so the parameters for HACMP and GPFS will have to be updated. Some of these changes are temporary, due to the reality of hardware interruptions, while others become permanent in order to achieve a desired goal.

7.1 GPFS administration

A properly configured /.rhost file must exist on every node of the GPFS cluster. This requirement only exists for the non-SP, or GPFS, cluster environment.

7.1.1 Managing the GPFS cluster

Some of the commands are dependent on the success of previous operations. Wherever possible, we step completely through each process.

GPFS cluster information

To list the current configuration for the cluster, run **mmlscluster**, as shown in Example 7-1. This command displays the current GPFS system data repositories (GSD), unassigned nodes and nodes within a nodeset.

Example 7-1 List GPFS cluster information

```
host1t:/> mmlscluster

GPFS cluster information
========================

GPFS system data repository servers:
------------------------------------
  Primary server:    host1e
  Secondary server:  host2e

Nodes for nodeset set1:
-----------------------------------------------
  1  host1e  129.40.12.129  host1e
  2  host2e  129.40.12.130  host2e
  3  host3e  129.40.12.131  host3e
  4  host4e  129.40.12.132  host4e
```

Deleting nodes from a cluster

Removing a node from a GPFS cluster first requires us to delete the node from the nodeset. This procedure, shown in Example 7-2 on page 111, is one of the more involved ones since the mmfs subsystem also has to be stopped and **mmdelnode -c** has to be run to adjust the list of cluster nodes in the GPFS configuration data. We have to stop **mmfs** on all nodes in the cluster whenever the -c option is specified. The node we delete will not be the primary or secondary GPFS system data repository server.

Order of events

- **mmshutdown** everywhere (does an automatic **umount** of the file system)
- **mmdelnode** the node to remove
- **mmdelcluster** the node to remove
- **mmstartup** to restore the mmfs subsystem to active
- **mount** the file system

Example 7-2 Deleting nodes from a cluster

```
host1t:/> mmshutdown -a
Thu Feb 22 11:52:16 EST 2001: mmshutdown: Starting force unmount of GPFS
filesystems
host1e:  forced unmount of /gpfs1
host2e:  forced unmount of /gpfs1
host4e:  forced unmount of /gpfs1
host3e:  forced unmount of /gpfs1
Thu Feb 22 11:52:21 EST 2001: mmshutdown: Shutting down GPFS daemons
host1e:  Shutting down!
host1e:  0513-044 The mmfs Subsystem was requested to stop.
host2e:  Shutting down!
host2e:  0513-044 The mmfs Subsystem was requested to stop.
host4e:  Shutting down!
host4e:  0513-044 The mmfs Subsystem was requested to stop.
host3e:  Shutting down!
host3e:  0513-044 The mmfs Subsystem was requested to stop.
Thu Feb 22 11:52:24 EST 2001: mmshutdown: Finished

host1t:/> mmdelnode -c -C set1 host3e
Verifying GPFS is stopped on all nodes ...
mmdelnode: Propagating the changes to all affected nodes.
This is an asynchronous process.

host1t:/> mmlscluster

GPFS cluster information
========================

GPFS system data repository servers:
------------------------------------
  Primary server:    host1e
  Secondary server:  host2e

Nodes for nodeset set1:
-----------------------------------------------
  1  host1e  129.40.12.129  host1e
  2  host2e  129.40.12.130  host2e
  4  host4e  129.40.12.132  host4e

Cluster nodes that are not assigned to a nodeset:
```

```
-------------------------------------------------
  3  host3e  129.40.12.131  host3e

host1t:/> mmlsnode -a
GPFS nodeset    Node list
-------------   -------------------------------------------------------
   set1         host1e host2e host4e

host1t:/> mmdelcluster host3e
mmdelcluster: Propagating the changes to all affected nodes.
This is an asynchronous process.
host1t:/> mmlscluster

GPFS cluster information
========================

GPFS system data repository servers:
-----------------------------------
  Primary server:    host1e
  Secondary server:  host2e

Nodes for nodeset set1:
-------------------------------------------------
  1  host1e  129.40.12.129  host1e
  2  host2e  129.40.12.130  host2e
  4  host4e  129.40.12.132  host4e

host1t:/> mmstartup -a
Wed Feb 21 17:38:26 EST 2001: mmstartup: Starting GPFS ...
host1e:  0513-059 The mmfs Subsystem has been started. Subsystem PID is 17268.
host2e:  0513-059 The mmfs Subsystem has been started. Subsystem PID is 18092.
host4e:  0513-059 The mmfs Subsystem has been started. Subsystem PID is 18442.

host1t:/> gdsh "ps -ef | grep mmfs"
host1t:     root 33300  3462   0 16:12:47      -  0:01 /usr/lpp/mmfs/bin/mmfsd
host2t:     root 11622  3886   0 16:17:33      -  0:02 /usr/lpp/mmfs/bin/mmfsd
host4t:     root 16730  3118   0 16:27:01      -  0:01 /usr/lpp/mmfs/bin/mmfsd

host1t:/> gdsh “mount /gpfs1”

host1t:/> gdsh "df -k | grep gpfs"
host1t: /dev/gpfs1      142077952 138653184    3%       13     1% /gpfs1
host2t: /dev/gpfs1      142077952 138653184    3%       13     1% /gpfs1
host4t: /dev/gpfs1      142077952 138653184    3%       13     1% /gpfs1

host2t:/> mmfsadm dump cfgmgr
Cluster Configuration: Type: 'HACMP'
Domain ,  3 nodes in this cluster
```

```
autoSgLoadBalance off

     node                adapter     admin          fails   SGs  mem daem TMreq
 idx   no host name      ip address   node  status panics mngd free CPU   /sec
---- ---- -------------- ----------- ----- ------- ------- --- ---- ---- -----
   0    1 host1e         129.40.12.129   y      up   0/0     1  99%   0%     0
   1    2 host2e         129.40.12.130   y      up   0/0     0  99%   0%     0
   2    4 host4e         129.40.12.132   y      up   0/0     0  99%   0%     0
```

Adding nodes to a cluster

To add a node to a cluster, that node cannot belong to another cluster, it must be available and a properly configured member of our HACMP/ES cluster. The steps in this proceedure are shown in Example 7-3.

Order of events

- **mmaddcluster** to add node to cluster
- **mmaddnode** to add node to specified nodeset
- **mount** to mount the file system

Example 7-3 Adding nodes to a cluster

```
host1t:/> mmlscluster

GPFS cluster information
========================

GPFS system data repository servers:
------------------------------------
  Primary server:    host1e
  Secondary server:  host2e

Nodes for nodeset set1:
-----------------------------------------------
  1  host1e  129.40.12.129  host1e
  2  host2e  129.40.12.130  host2e
  4  host4e  129.40.12.132  host4e

host1t:/> mmaddcluster host3e
mmaddcluster: Command successfully completed
host1t:/> mmlscluster

GPFS cluster information
========================

GPFS system data respository servers:
-------------------------------------
```

```
  Primary server:    host1e
  Secondary server:  host2e

Nodes for nodeset set1:
------------------------------------------------
  1  host1e  129.40.12.129  host1e
  2  host2e  129.40.12.130  host2e
  4  host4e  129.40.12.132  host4e

Cluster nodes that are not assigned to a nodeset:
------------------------------------------------
  3  host3e  129.40.12.131  host3e

host1t:/> mmaddnode -C set1 host3e
mmaddnode: Propagating the changes to all affected nodes.
This is an asynchronous process.

host3t:/> mmlsnode -a
GPFS nodeset    Node list
-------------   -------------------------------------------------------
   set1         host1e host2e host4e host3e

host1t:/> mmlscluster

GPFS cluster information
========================

GPFS system data repository servers:
-----------------------------------
  Primary server:    host1e
  Secondary server:  host2e

Nodes for nodeset set1:
------------------------------------------------
  1  host1e  129.40.12.129  host1e
  2  host2e  129.40.12.130  host2e
  4  host4e  129.40.12.132  host4e
  3  host3e  129.40.12.131  host3e

host2t:/> gdsh -w host3t "startsrc -s mmfs"
host3t: 0513-059 The mmfs Subsystem has been started. Subsystem PID is 5006.

host2t:/> mmfsadm dump cfgmgr
Cluster Configuration: Type: 'HACMP'
Domain ,  4 nodes in this cluster
    - 3 nodes (Used in quorum calculation)
    - 1 new node (Not used in quorum calculation)

autoSgLoadBalance off
```

```
     node                adapter     admin         fails   SGs  mem daem TMreq
 idx   no host name      ip address   node  status panics mngd free CPU   /sec
---- ---- -------------- ----------- ----- ------- ------- --- ---- ---- -----
   0    1 host1e         129.40.12.129   y       up   0/0     1  99%   0%     0
   1    2 host2e         129.40.12.130   y       up   0/0     0  99%   0%     0
   2    4 host4e         129.40.12.132   y       up   0/0     0  99%   0%     0

New node list:

     node                adapter     admin         fails   SGs  mem daem TMreq
 idx   no host name      ip address   node  status panics mngd free CPU   /sec
---- ---- -------------- ----------- ----- ------- ------- --- ---- ---- -----
   3    3 host3e         129.40.12.131   y    down   0/0     0   0% 100%

After giving node host3e a minute to complete dynamic addnode processing, mount
the file system.

host2t:/> gdsh -w host3t "mount /gpfs1"

host2t:/> mmfsadm dump cfgmgr
Cluster Configuration: Type: 'HACMP'
Domain ,  4 nodes in this cluster

autoSgLoadBalance off

     node                adapter     admin         fails   SGs  mem daem TMreq
 idx   no host name      ip address   node  status panics mngd free CPU   /sec
---- ---- -------------- ----------- ----- ------- ------- --- ---- ---- -----
   0    1 host1e         129.40.12.129   y       up   0/0     1  99%   0%     0
   1    2 host2e         129.40.12.130   y       up   0/0     0  99%   0%     0
   2    4 host4e         129.40.12.132   y       up   0/0     0  99%   0%     0
   3    3 host3e         129.40.12.131   y       up   0/0     0  99%   0%     0

host3t:/> cat /var/adm/ras/mmfs.log.latest
Thu Feb 22 11:09:36 EST 2001 /usr/lpp/mmfs/bin/runmmfs starting
Removing old /var/adm/ras/mmfs.log.* files:
/usr/lpp/mmfs/bin/mmfskxload: /usr/lpp/mmfs/bin/mmfs is already loaded at
897819
64.
Thu Feb 22 11:09:39 2001: mmfsd initializing. {Version: 3.3.0.0   Built: Feb  2
2001 16:44:59} ...
Thu Feb 22 11:09:40 2001: useSPSecurity no
Thu Feb 22 11:09:40 2001: Cluster type: 'HACMP'
Thu Feb 22 11:10:07 2001: Using TCP communication protocol
/usr/bin/lslpp: Fileset bos.up not installed.
Thu Feb 22 11:10:07 2001: Connecting to 129.40.12.130
Thu Feb 22 11:10:07 2001: Connected to 129.40.12.130
Thu Feb 22 11:10:07 EST 2001 /var/mmfs/etc/gpfsready invoked
```

```
Thu Feb 22 11:10:07 2001: mmfsd ready
Thu Feb 22 11:11:06 2001: Command: mount /dev/gpfs1 17412
Thu Feb 22 11:11:06 2001: Connecting to 129.40.12.129
Thu Feb 22 11:11:06 2001: Connected to 129.40.12.129
Activated gpfsvg0 successfully.
Activated gpfsvg1 successfully.
Activated gpfsvg2 successfully.
Activated gpfsvg3 successfully.
Activated gpfsvg4 successfully.
Activated gpfsvg5 successfully.
Activated gpfsvg6 successfully.
Activated gpfsvg7 successfully.
Activated gpfsvg8 successfully.
Activated gpfsvg9 successfully.
Activated gpfsvg10 successfully.
Activated gpfsvg11 successfully.
Activated gpfsvg12 successfully.
Activated gpfsvg13 successfully.
Activated gpfsvg14 successfully.
Activated gpfsvg15 successfully.

host3t:/> gdsh "df -k | grep gpfs"
host1t: /dev/gpfs1      142077952 138653184    3%       13     1% /gpfs1
host2t: /dev/gpfs1      142077952 138653184    3%       13     1% /gpfs1
host3t: /dev/gpfs1      142077952 138653184    3%       13     1% /gpfs1
host4t: /dev/gpfs1      142077952 138653184    3%       13     1% /gpfs1
```

Changing the system data server

When we set up our initial GPFS cluster, we designated a primary and secondary server. If for any reason we wanted to remove the primary server, we would move that responsibility to another node using the **mmchcluster** command first, as shown in Example 7-4.

Example 7-4 Changing the primary GPFS cluster

```
host3t:/> mmlscluster

GPFS cluster information
========================

GPFS system data respository servers:
-------------------------------------
  Primary server:    host1e
  Secondary server:  host2e

Nodes for nodeset set1:
-----------------------------------------------
  1  host1e  129.40.12.129  host1e
```

```
  2  host2e  129.40.12.130  host2e
  4  host4e  129.40.12.132  host4e
  3  host3e  129.40.12.131  host3e


host3t:/> mmchcluster -p host3e
mmchcluster: Command successfully completed
host3t:/> mmlscluster

GPFS cluster information
========================

GPFS system data respository servers:
-------------------------------------
  Primary server:    host3e
  Secondary server:  host2e

Nodes for nodeset set1:
-----------------------------------------------
  1  host1e  129.40.12.129  host1e
  2  host2e  129.40.12.130  host2e
  4  host4e  129.40.12.132  host4e
  3  host3e  129.40.12.131  host3e
```

7.1.2 Managing the GPFS configuration

We could tune our GPFS configuration after is was up and running by using **mmconfig**. As shown in Example 7-5, some of the attributes that are alterable are pagepool and autoload. These parameters can be changed immediately, be made permanent, or even not persist after GPFS is restarted.

Example 7-5 Changing the GPFS configuration

```
host1t:/> mmlsconfig
Configuration data for nodeset set1:
------------------------------------
pagepool 80M
dataStructureDump /tmp/mmfs
multinode yes
autoload no
useSingleNodeQuorum no
wait4RVSD no
comm_protocol TCP
clusterType hacmp
group Gpfs.set1
recgroup GpfsRec.set1
```

```
File systems in nodeset set1:
-----------------------------
/dev/gpfs1

host1t:/> mmchconfig pagepool=100M -C set1 -i
mmchconfig: Command successfully completed
mmchconfig: Propagating the changes to all affected nodes.
This is an asynchronous process.
host1t:/> mmlsconfig
Configuration data for nodeset set1:
------------------------------------
pagepool 100M
dataStructureDump /tmp/mmfs
multinode yes
autoload no
useSingleNodeQuorum no
wait4RVSD no
comm_protocol TCP
clusterType hacmp
group Gpfs.set1
recgroup GpfsRec.set1

File systems in nodeset set1:
-----------------------------
/dev/gpfs1

host1t:/> mmchconfig autoload=yes -C set1
mmchconfig: Command successfully completed
mmchconfig: Propagating the changes to all affected nodes.
This is an asynchronous process.
host1t:/> mmlsconfig
Configuration data for nodeset set1:
------------------------------------
pagepool 100M
dataStructureDump /tmp/mmfs
multinode yes
autoload yes
useSingleNodeQuorum no
wait4RVSD no
comm_protocol TCP
```

```
clusterType hacmp
group Gpfs.set1
recgroup GpfsRec.set1

File systems in nodeset set1:
-----------------------------
/dev/gpfs1
```

7.1.3 Unmounting and stopping GPFS

The command **mmshutdown** will stop GPFS and will also umount the file system. An alternative command would be to **umount /gpfs1** on all the nodes and then **stopsrc -s mmfs** on all the affected nodes. This procedure is more cumbersome than the **mmshutdown** command, as shown in Example 7-6.

Example 7-6 Stopping GPFS

```
host1t:/> mmshutdown -C set1
Thu Feb 22 11:52:16 EST 2001: mmshutdown: Starting force unmount of GPFS
filesystems
host1e:  forced unmount of /gpfs1
host2e:  forced unmount of /gpfs1
host4e:  forced unmount of /gpfs1
host3e:  forced unmount of /gpfs1
Thu Feb 22 11:52:21 EST 2001: mmshutdown: Shutting down GPFS daemons
host1e:  Shutting down!
host1e:  0513-044 The mmfs Subsystem was requested to stop.
host2e:  Shutting down!
host2e:  0513-044 The mmfs Subsystem was requested to stop.
host4e:  Shutting down!
host4e:  0513-044 The mmfs Subsystem was requested to stop.
host3e:  Shutting down!
host3e:  0513-044 The mmfs Subsystem was requested to stop.
Thu Feb 22 11:52:24 EST 2001: mmshutdown: Finished

host1t:/> gdsh "lssrc -s mmfs"
host1t: Subsystem         Group            PID     Status
host1t:  mmfs             aixmm                    inoperative
host2t: Subsystem         Group            PID     Status
host2t:  mmfs             aixmm                    inoperative
host3t: Subsystem         Group            PID     Status
host3t:  mmfs             aixmm                    inoperative
host4t: Subsystem         Group            PID     Status
host4t:  mmfs             aixmm                    inoperative

host1t:/> cat /var/adm/ras/mmfs.log.latest
Thu Feb 22 11:52:22 2001: mmfsd shutting down.
Thu Feb 22 11:52:22 2001: Reason for shutdown: Normal shutdown
```

```
/var/mmfs/etc/mmfsdown.scr: /usr/bin/lssrc -s mmfs
Subsystem         Group            PID     Status
 mmfs             aixmm            17272   stopping
/var/mmfs/etc/mmfsdown.scr: /usr/sbin/umount -f -t mmfs
```

7.1.4 Starting and mounting GPFS

The mmfs subsystem has to be started and then the file system has to be mounted, as shown in Example 7-7. The command **mmstartup -C NodesetId** will start the subsystem on all nodes in the specified nodeset. An alternate method would be to use **startsrc -s mmfs** on all desired nodes. Had we designated in the mmconfig command that Autoload=yes (viewable with **mmlsconfig**), this step would be unnecessary during a reboot as **mmfs** would start automatically.

The next step is to mount the file system with the **mount /gpfs1** command on every desired node. Had we designated Automatic mount option=yes during the **mmcrfs** command (now viewable by **mmlfs gpfs1**) the file system would mount whenever mmfs became available.

Example 7-7 Start GPFS

```
host1t:/> mmstartup -C set1
Thu Feb 22 11:55:39 EST 2001: mmstartup: Starting GPFS ...
host1e:  0513-059 The mmfs Subsystem has been started. Subsystem PID is 31800.
host2e:  0513-059 The mmfs Subsystem has been started. Subsystem PID is 18096.
host4e:  0513-059 The mmfs Subsystem has been started. Subsystem PID is 12210.
host3e:  0513-059 The mmfs Subsystem has been started. Subsystem PID is 5008.

host1t:/> mmfsadm dump cfgmgr
Cluster Configuration: Type: 'HACMP'
Domain ,  4 nodes in this cluster

autoSgLoadBalance off

     node                adapter     admin         fails   SGs  mem daem TMreq
 idx   no host name      ip address   node  status panics mngd free CPU   /sec
---- ---- -------------- ----------- ----- ------- ------- --- ---- ---- -----
   0    1 host1e         129.40.12.129   y       up   0/0     0   0% 100%     0
   1    2 host2e         129.40.12.130   y       up   0/0     0   0% 100%     0
   2    4 host4e         129.40.12.132   y       up   0/0     0   0% 100%     0
   3    3 host3e         129.40.12.131   y       up   0/0     0   0% 100%     0

Cluster configuration manager is 129.40.12.130 (other node)
Stripe groups managed by this node:
  (none)
Using TCP communication with epoch number 0, PrevLapiEpochNo 0
Phoenix Group Names:Gpfs.set1, GpfsRec.set1; Group State:Active; Quorum:3
Versio
```

```
n:(5:5)

host1t:/> gdsh "lssrc -s mmfs"
host1t: Subsystem         Group            PID     Status
host1t:  mmfs             aixmm            31800   active
host2t: Subsystem         Group            PID     Status
host2t:  mmfs             aixmm            18096   active
host3t: Subsystem         Group            PID     Status
host3t:  mmfs             aixmm            5008    active
host4t: Subsystem         Group            PID     Status
host4t:  mmfs             aixmm            12210   active

host1t:/> cat /var/adm/ras/mmfs.log.latest
Thu Feb 22 11:55:43 EST 2001 /usr/lpp/mmfs/bin/runmmfs starting
Removing old /var/adm/ras/mmfs.log.* files:
/usr/lpp/mmfs/bin/mmfskxload: /usr/lpp/mmfs/bin/mmfs is already loaded at
897123
32.
Thu Feb 22 11:55:46 2001: mmfsd initializing. {Version: 3.3.0.0   Built: Feb  2
2001 16:44:59} ...
Thu Feb 22 11:55:47 2001: useSPSecurity no
Thu Feb 22 11:55:48 2001: Cluster type: 'HACMP'
Thu Feb 22 11:55:48 2001: Using TCP communication protocol
/usr/bin/lslpp: Fileset bos.up not installed.
Thu Feb 22 11:55:48 2001: Connecting to 129.40.12.130
Thu Feb 22 11:55:48 2001: Connected to 129.40.12.130
Thu Feb 22 11:55:48 EST 2001 /var/mmfs/etc/gpfsready invoked
Thu Feb 22 11:55:48 2001: mmfsd ready
Thu Feb 22 11:56:04 2001: Command: dump cfgmgr

host1t:/> gdsh "mount /gpfs1"

host1t:/> gdsh "df -k | grep gpfs"
host1t: /dev/gpfs1      142077952 138653184    3%       13     1% /gpfs1
host2t: /dev/gpfs1      142077952 138653184    3%       13     1% /gpfs1
host3t: /dev/gpfs1      142077952 138653184    3%       13     1% /gpfs1
host4t: /dev/gpfs1      142077952 138653184    3%       13     1% /gpfs1

host1t:/> tail /var/adm/ras/mmfs.log.latest
Thu Feb 22 11:57:55 2001: Command: mount /dev/gpfs1 28744
Thu Feb 22 11:58:12 2001: Accepted and connected to 129.40.12.131
Thu Feb 22 11:58:15 2001: Accepted and connected to 129.40.12.132
```

7.1.5 Managing the file system

Some of the following command and techniques are mentioned elsewhere in this section since other topics are dependent on them. We'll cover them briefly here along with any new topics in file system management.

Mounting the file system

Once the mmfs subsystem is up and running, we issued the **mount /gpfs1** command, as shown in Example 7-8, to all the nodes where we wanted it mounted.

Example 7-8 Mount GPFA file system

```
host1t:/> gdsh "mount /gpfs1"

host1t:/> gdsh "df -k | grep gpfs"
host1t: /dev/gpfs1      142077952 138653184    3%       13     1% /gpfs1
host2t: /dev/gpfs1      142077952 138653184    3%       13     1% /gpfs1
host3t: /dev/gpfs1      142077952 138653184    3%       13     1% /gpfs1
host4t: /dev/gpfs1      142077952 138653184    3%       13     1% /gpfs1
```

Unmounting the file system

In Example 7-9, we unmounted the file system by issuing **umount /gpfs1** to all the appropriate nodes. If we also want to bring down GPFS, we could use **mmshutdown -C set1**, which automatically unmounts the file system and stops the mmfs subsystem.

Example 7-9 Unmounting the GPFS file system

```
host3t:/> gdsh “umount /gpfs1”
host1t:/> gdsh "df -k | grep gpfs"
host1t:/>

host1t:/> mmshutdown -C set1
Thu Feb 22 11:52:16 EST 2001: mmshutdown: Starting force unmount of GPFS
filesys
tems
host1e:  forced unmount of /gpfs1
host2e:  forced unmount of /gpfs1
host4e:  forced unmount of /gpfs1
host3e:  forced unmount of /gpfs1
Thu Feb 22 11:52:21 EST 2001: mmshutdown: Shutting down GPFS daemons
host1e:  Shutting down!
host1e:  0513-044 The mmfs Subsystem was requested to stop.
host2e:  Shutting down!
host2e:  0513-044 The mmfs Subsystem was requested to stop.
host4e:  Shutting down!
host4e:  0513-044 The mmfs Subsystem was requested to stop.
```

```
host3e:  Shutting down!
host3e:  0513-044 The mmfs Subsystem was requested to stop.
Thu Feb 22 11:52:24 EST 2001: mmshutdown: Finished
```

Deleting the file system

Before deleting a file system, unmount it on all nodes and then run **mmdelfs** from any node, as shown in Example 7-10.

Example 7-10 Delete the GPFS file system

```
host1t:/> mmdelfs gpfs1


mmdelfs: 6027-1366 Marking the disks as available
GPFS: 6027-573 All data on following disks of gpfs1 will be destroyed:
gpfslv0
gpfslv1
gpfslv2
gpfslv3
gpfslv4
gpfslv5
gpfslv6
gpfslv7
gpfslv8
gpfslv9
gpfslv10
gpfslv11
gpfslv12
gpfslv13
gpfslv14
gpfslv15
GPFS: 6027-574 Completed deletion of file system gpfs1.
mmdelfs: 6027-1371 Propagating the changes to all affected nodes.
This is an asynchronous process.
```

Checking and repairing the file system

The **mmfsck** command finds and repairs problem conditions in the file system. This command can be run in two modes, online or offline. Online mode will check and recover unallocated blocks on the mounted file system. Offline mode checks for file inconsistencies in an unmounted file system. Before the command can be initiated, run **mmlsdisk**, as shown in Example 7-11 on page 124, and make sure none of the disks are in a down state. Run **mmchdisk** to change the state of the disk to unrecovered or up.

Example 7-11 Check and repair GPFS file system

```
host1t:/> mmfsck gpfs1
Cannot check. "gpfs1" is mounted on 4 node(s) and in use on 4 node(s).
host1t:/> gdsh "umount /gpfs1"
host1t:/> mmfsck gpfs1
Checking "gpfs1"
Checking inodes
Checking inode map file
Checking directories and files
Checking log files
Checking extended attributes file
Checking file reference counts
Checking file system replication status
139264 inodes
          17   allocated
           0   repairable
           0   repaired
           0   damaged
           0   deallocated
           0   orphaned
           0   attached

    17760256 subblocks
      428709   allocated
           0   unreferenced
           0   deletable
           0   deallocated

File system is clean.
```

Listing file system attributes

The best command for checking the file system attributes is **mmlsfs**, as shown in Example 7-12.

Example 7-12 List file system attributes

```
host1t:/> mmlsfs gpfs1
flag value          description
---- -------------- -----------------------------------------------------
 -s  roundRobin     Stripe method
 -f  8192           Minimum fragment size in bytes
 -i  512            Inode size in bytes
 -I  16384          Indirect block size in bytes
 -m  1              Default number of metadata replicas
 -M  1              Maximum number of metadata replicas
 -r  1              Default number of data replicas
 -R  1              Maximum number of data replicas
 -a  1048576        Estimated average file size
```

```
 -n  32             Estimated number of nodes that will mount file system
 -B  262144         Block size
 -Q  none           Quotas enforced
 -F  139264         Maximum number of inodes
 -V  4              File system version. Highest supported version: 4
 -z  no             Is DMAPI enabled?
 -d
gpfslv0;gpfslv1;gpfslv2;gpfslv3;gpfslv4;gpfslv5;gpfslv6;gpfslv7;gpfslv8;gpf
slv9;gpfslv10;gpfslv11;gpfslv12;gpfslv13;gpfslv14;gpfslv15  Disks in file
system
 -A  no             Automatic mount option
 -C  set1           GPFS nodeset identifier
 -E  no             Exact mtime default mount option
 -S  no             Suppress atime default mount option
```

Modifying file system attributes

The **mmchfs** command performs the function of modifying existing file system attributes. Current attributes are listed by running **mmlsfs**.

Balancing data with mmdf and mmrestripefs

Data is evenly striped across all active disks in the file system. When a disk is deleted, the data should be spread evenly across all remaining disks. When a disk is added to the file system, it will appear with zero data unless we specifically request that data be rebalanced across all disks. To view the existing data ratios run **mmdf**; to rebalance the data run **mmrestripefs**, as shown in Example 7-13. To make sure all the disks we want to balance across are available, we run **mmlsdisk**. Running **mmchdisk** allowed us to change the disk status so it was or was not affected by the **mmrestripefs** command.

Example 7-13 View data ratios and redistribute

```
host1t:/> mmdf gpfs1
disk            disk size  failure holds    holds         free KB         free KB
name                in KB    group metadata data   in full blocks    in fragments
--------------- --------- -------- -------- ----- --------------- ---------------
gpfslv0           8880128        1 yes      yes     8665088 (98%)       552 ( 0%)
gpfslv1           8880128        1 yes      yes     8665088 (98%)       552 ( 0%)
gpfslv2           8880128        1 yes      yes     8665856 (98%)       312 ( 0%)
gpfslv3           8880128        1 yes      yes     8665344 (98%)       560 ( 0%)
gpfslv4           8880128        1 yes      yes     8665344 (98%)       296 ( 0%)
gpfslv5           8880128        1 yes      yes     8665344 (98%)       312 ( 0%)
gpfslv6           8880128        1 yes      yes     8665600 (98%)       312 ( 0%)
gpfslv7           8880128        1 yes      yes     8664320 (98%)       552 ( 0%)
gpfslv8           8880128        1 yes      yes     8665088 (98%)       568 ( 0%)
gpfslv9           8880128        1 yes      yes     8665344 (98%)       568 ( 0%)
gpfslv10          8880128        1 yes      yes     8665344 (98%)       568 ( 0%)
gpfslv11          8880128        1 yes      yes     8665088 (98%)       808 ( 0%)
```

```
gpfslv12          8880128        1 yes      yes     8665088 (98%)       808 ( 0%)
gpfslv13          8880128        1 yes      yes     8665344 (98%)       552 ( 0%)
gpfslv14          8880128        1 yes      yes     8665088 (98%)       792 ( 0%)
gpfslv15          8880128        1 yes      yes     8665344 (98%)       552 ( 0%)
                 ---------                         -------------- --------------
(total)         142082048                        138643712 (98%)      8664 ( 0%)
Inode Information
-----------------
Total number of inodes: 139264
Total number of free inodes: 139247

host1t:/> mmrestripefs gpfs1 -r
Scanning file system metadata, phase 1 ...
Scan completed successfully.
Scanning file system metadata, phase 2 ...
Scan completed successfully.
Scanning file system metadata, phase 3 ...
Scan completed successfully.
Scanning user file metadata ...
Scan completed successfully.
```

File system fragmentation and defragmentation

When a file is closed after it has written the last logical block of data, that data is reduced to the actual number of subblocks required, thus creating a fragmented block. The **mmdefragfs** command, as shown in Example 7-14, can be used to query the current fragmented state of the file system and to reduce the fragmentation of the file system.

Example 7-14 Defragment file system

```
host1t:/tools/ralph> mmdefragfs gpfs1
Warning: "gpfs1" is mounted  on 4 node(s) and in use on 4 node(s)
Start processing: iteration 1
Processing Pass 1 of 1
Disk Name: gpfslv0 gpfslv1 gpfslv2 gpfslv3 gpfslv4 gpfslv5 gpfslv6 gpfslv7
gpfsl
v8 gpfslv9 gpfslv10 gpfslv11 gpfslv12 gpfslv13 gpfslv14 gpfslv15
  41 % complete on Fri Feb 23 17:58:58 2001
  63 % complete on Fri Feb 23 17:59:03 2001
  83 % complete on Fri Feb 23 17:59:08 2001

               free subblk              free
disk            in full               subblk in          %             %
name            blocks        blk     fragments        free blk     blk util
             before    after freed   before    after before after before after
---------- ---------- ---------- ---------- ---------- ------------ ------------
gpfslv0      519808   519808     0      219      209  46.83 46.83  99.96 99.96
gpfslv1      519776   519776     0      279      265  46.83 46.83  99.95 99.96
```

```
gpfslv2      519904   519904     0      219      209  46.84 46.84  99.96 99.96
gpfslv3      519840   519840     0      250      240  46.83 46.83  99.96 99.96
gpfslv4      519776   519776     0      277      263  46.83 46.83  99.95 99.96
gpfslv5      519840   519840     0      219      209  46.83 46.83  99.96 99.96
gpfslv6      519872   519872     0      219      209  46.83 46.83  99.96 99.96
gpfslv7      519712   519712     0      249      237  46.82 46.82  99.96 99.96
gpfslv8      519744   519744     0      339      325  46.82 46.82  99.94 99.94
gpfslv9      519872   519872     0      311      297  46.83 46.83  99.95 99.95
gpfslv10     519840   519840     0      371      353  46.83 46.83  99.94 99.94
gpfslv11     519808   519808     0      371      353  46.83 46.83  99.94 99.94
gpfslv12     519840   519840     0      311      297  46.83 46.83  99.95 99.95
gpfslv13     519776   519776     0      339      321  46.83 46.83  99.94 99.95
gpfslv14     519744   519744     0      337      323  46.82 46.82  99.94 99.95
gpfslv15     519744   519744     0      309      297  46.82 46.82  99.95 99.95
             ---------------------- -----------------              ------------
(total)     8316896  8316896     0     4619     4407                99.95 99.95
```

7.1.6 Managing disks

Disk drives in a file system are the most likely item to be replaced, either due to mechanical failure or upgrade. Removal, addition, or even swapping of a disk drive can be done with the file system mounted and online. In a cluster environment, a properly configured /.rhost file must exist on each node in the GPFS cluster. Since GPFS remained operational during the delete, replace, and add disk operations, all that activity was captured in /var/adm/ras/mmfs.log.latest for later referral.

Deleting disks from a file system

First we view the current status of all the disks in the file system with the **mmlsdisk gpfs1** command, as shown in Example 7-15.

Example 7-15 View status of disks

```
host1t:/> mmlsdisk gpfs1
disk         driver   sector failure holds    holds
name         type       size   group metadata data  status        availability
------------ -------- ------ ------- -------- ----- ------------- ------------
gpfslv0      disk        512       1 yes      yes   ready         up
gpfslv1      disk        512       1 yes      yes   ready         up
gpfslv2      disk        512       1 yes      yes   ready         up
gpfslv3      disk        512       1 yes      yes   ready         up
gpfslv4      disk        512       1 yes      yes   ready         up
gpfslv5      disk        512       1 yes      yes   ready         up
gpfslv6      disk        512       1 yes      yes   ready         up
gpfslv7      disk        512       1 yes      yes   ready         up
gpfslv8      disk        512       1 yes      yes   ready         up
gpfslv9      disk        512       1 yes      yes   ready         up
```

```
gpfslv10     disk        512       1 yes      yes   ready         up
gpfslv11     disk        512       1 yes      yes   ready         up
gpfslv12     disk        512       1 yes      yes   ready         up
gpfslv14     disk        512       1 yes      yes   ready         up
gpfslv15     disk        512       1 yes      yes   ready         up
```

Before a disk can be replaced, we must suspend the disk. By using the **mmchdisk** command, as shown in Example 7-16, a disk can be placed in several states, such as suspend, start, and stop. You cannot write new data to a suspended disk but you can update existing data and read from that disk. A disk is typically suspended prior to restriping or deletion. We arbitrarily chose to remove gpfslv13 for our test.

Example 7-16 Change disk status to suspend

```
host1t:/> mmchdisk gpfs1 suspend -d gpfslv13
host1t:/> mmlsdisk gpfs1
disk         driver   sector failure holds    holds
name         type       size   group metadata data  status        availability
------------ -------- ------ ------- -------- ----- ------------- ------------
gpfslv0      disk        512       1 yes      yes   ready         up
gpfslv1      disk        512       1 yes      yes   ready         up
gpfslv2      disk        512       1 yes      yes   ready         up
gpfslv3      disk        512       1 yes      yes   ready         up
gpfslv4      disk        512       1 yes      yes   ready         up
gpfslv5      disk        512       1 yes      yes   ready         up
gpfslv6      disk        512       1 yes      yes   ready         up
gpfslv7      disk        512       1 yes      yes   ready         up
gpfslv8      disk        512       1 yes      yes   ready         up
gpfslv9      disk        512       1 yes      yes   ready         up
gpfslv10     disk        512       1 yes      yes   ready         up
gpfslv11     disk        512       1 yes      yes   ready         up
gpfslv12     disk        512       1 yes      yes   ready         up
gpfslv13     disk        512       1 yes      yes   suspended     up
gpfslv14     disk        512       1 yes      yes   ready         up
gpfslv15     disk        512       1 yes      yes   ready         up
```

Now that we have observed gpfslv13 is suspended we could immediately delete it with **mmdeldisk**, as shown in Example 7-17. We used this command with a **-r** modifier to restripe the data across the remaining disks, rebalancing the data. Notice that the **mmlsdisk** command reveals that gpfslv13 is now missing.

Example 7-17 Delete disk

```
host1t:/> mmdeldisk gpfs1 gpfslv13 -r
Deleting disks ...
Scanning file system metadata, phase 1 ...
Scan completed successfully.
Scanning file system metadata, phase 2 ...
```

```
Scan completed successfully.
Scanning file system metadata, phase 3 ...
Scan completed successfully.
Scanning user file metadata ...
 100 % complete on Thu Feb 22 13:51:13 2001
Scan completed successfully.
tsdeldisk completed.
mmdeldisk: Propagating the changes to all affected nodes.
This is an asynchronous process.
Restriping gpfs1 ...
Scanning file system metadata, phase 1 ...
  23 % complete on Thu Feb 22 13:51:23 2001
  45 % complete on Thu Feb 22 13:51:26 2001
  68 % complete on Thu Feb 22 13:51:29 2001
  90 % complete on Thu Feb 22 13:51:32 2001
 100 % complete on Thu Feb 22 13:51:33 2001
Scan completed successfully.
Scanning file system metadata, phase 2 ...
Scan completed successfully.
Scanning file system metadata, phase 3 ...
Scan completed successfully.
Scanning user file metadata ...
 100 % complete on Thu Feb 22 14:01:28 2001
Scan completed successfully.
Done

host1t:/> mmlsdisk gpfs1
disk         driver   sector failure holds    holds
name         type       size   group metadata data  status        availability
------------ -------- ------ ------- -------- ----- ------------- ------------
gpfslv0      disk        512       1 yes      yes   ready         up
gpfslv1      disk        512       1 yes      yes   ready         up
gpfslv2      disk        512       1 yes      yes   ready         up
gpfslv3      disk        512       1 yes      yes   ready         up
gpfslv4      disk        512       1 yes      yes   ready         up
gpfslv5      disk        512       1 yes      yes   ready         up
gpfslv6      disk        512       1 yes      yes   ready         up
gpfslv7      disk        512       1 yes      yes   ready         up
gpfslv8      disk        512       1 yes      yes   ready         up
gpfslv9      disk        512       1 yes      yes   ready         up
gpfslv10     disk        512       1 yes      yes   ready         up
gpfslv11     disk        512       1 yes      yes   ready         up
gpfslv12     disk        512       1 yes      yes   ready         up
gpfslv14     disk        512       1 yes      yes   ready         up
gpfslv15     disk        512       1 yes      yes   ready         up
```

Replacing disks in a file system

Replacing an existing disk in a GPFS file system with a new one is the same as performing a delete disk operation followed by an add disk operation. The benefit in doing it in one step is that the data does not need to be redistributed. A simple automated copy to the new disk and it should look just like before the command was issued. The new disk has to have a volume group and logical volume name assigned to it and it has to be a concurrent volume, see Chapter 6, “Configuring GPFS and SSA disks” on page 91 for a more detailed information.

In Example 7-18, we replaced gpfslv7 with the earlier deleted disk gpfslv13. Note that concurrent mode is capable, an absolute in the case of a non-VSD environment. The disk to be replaced is in the ready status mode, we do not place that disk in suspend mode for a **mmrpldisk** operation. Notice in the **mmlsdisk** example that gpfslv7 is now missing while gpfslv13 is again present.

Example 7-18 Replacing disks in file system

```
host1t:/gpfs1> lspv hdisk16
PHYSICAL VOLUME:    hdisk16                  VOLUME GROUP:     gpfsvg13
PV IDENTIFIER:      000007024db52d7b0000000000000000 VG IDENTIFIER
000b4a7d7
0d7eecd
PV STATE:           active
STALE PARTITIONS:   0                        ALLOCATABLE:      yes
PP SIZE:            16 megabyte(s)           LOGICAL VOLUMES:  1
TOTAL PPs:          542 (8672 megabytes)     VG DESCRIPTORS:   2
FREE PPs:           0 (0 megabytes)
USED PPs:           542 (8672 megabytes)
FREE DISTRIBUTION:  00..00..00..00..00
USED DISTRIBUTION:  109..108..108..108..109

host1t:/gpfs1> lsvg gpfsvg13
VOLUME GROUP:   gpfsvg13                 VG IDENTIFIER:  000b4a7d70d7eecd
VG STATE:       active                   PP SIZE:        16 megabyte(s)
VG PERMISSION:  read/write               TOTAL PPs:      542 (8672 megabytes)
MAX LVs:        256                      FREE PPs:       0 (0 megabytes)
LVs:            1                        USED PPs:       542 (8672 megabytes)
OPEN LVs:       0                        QUORUM:         2
TOTAL PVs:      1                        VG DESCRIPTORS: 2
STALE PVs:      0                        STALE PPs:      0
ACTIVE PVs:     1                        AUTO ON:        no
Concurrent:     Capable                  Auto-Concurrent: Disabled
VG Mode:        Non-Concurrent
MAX PPs per PV: 1016                     MAX PVs:        32

host1t:/gpfs1> mmrpldisk gpfs1 gpfslv7 gpfslv13::::1
Replacing gpfslv7 ...

The following disks of gpfs1 will be formatted on node host1t:
```

```
    gpfslv13: size 8880128 KB
Extending Allocation Map
Completed adding disks to file system gpfs1.
Scanning file system metadata, phase 1 ...
Scan completed successfully.
Scanning file system metadata, phase 2 ...
Scan completed successfully.
Scanning file system metadata, phase 3 ...
Scan completed successfully.
Scanning user file metadata ...
 100 % complete on Thu Feb 22 14:19:08 2001
Scan completed successfully.
Done
mmrpldisk: Propagating the changes to all affected nodes.
This is an asynchronous process.

host1t:/gpfs1> mmlsdisk gpfs1
disk         driver   sector failure holds    holds
name         type       size   group metadata data  status        availability
------------ -------- ------ ------- -------- ----- ------------- ------------
gpfslv0      disk        512       1 yes      yes   ready         up
gpfslv1      disk        512       1 yes      yes   ready         up
gpfslv2      disk        512       1 yes      yes   ready         up
gpfslv3      disk        512       1 yes      yes   ready         up
gpfslv4      disk        512       1 yes      yes   ready         up
gpfslv5      disk        512       1 yes      yes   ready         up
gpfslv6      disk        512       1 yes      yes   ready         up
gpfslv8      disk        512       1 yes      yes   ready         up
gpfslv9      disk        512       1 yes      yes   ready         up
gpfslv10     disk        512       1 yes      yes   ready         up
gpfslv11     disk        512       1 yes      yes   ready         up
gpfslv12     disk        512       1 yes      yes   ready         up
gpfslv13     disk        512       1 yes      yes   ready         up
gpfslv14     disk        512       1 yes      yes   ready         up
gpfslv15     disk        512       1 yes      yes   ready         up
```

Adding disks to a file system

Adding a disk requires the new disk to have a volume group name and a logical volume name (see Chapter 6, “Configuring GPFS and SSA disks” on page 91 for more information). In Example 7-19 on page 132, we used the recently replaced disk gpfslv7 as our new disk. We also used **mmadddisk -r** to redistribute or rebalance the data across all disks evenly instead of leaving the new disk with zero data. When specifying this new disk the failure group “1” must be explicitly named (`gpfslv7::::1`) or this disk will default to the “-1” failure group.

Example 7-19 Adding disks to a file system

```
st1t:/gpfs1> mmadddisk gpfs1 gpfslv7::::1 -r

The following disks of gpfs1 will be formatted on node host1t:
    gpfslv7: size 8880128 KB
Extending Allocation Map
Completed adding disks to file system gpfs1.
mmadddisk: Propagating the changes to all affected nodes.
This is an asynchronous process.
Restriping gpfs1 ...
Scanning file system metadata, phase 1 ...
  22 % complete on Thu Feb 22 14:24:53 2001
  41 % complete on Thu Feb 22 14:24:56 2001
  60 % complete on Thu Feb 22 14:24:59 2001
  79 % complete on Thu Feb 22 14:25:02 2001
 100 % complete on Thu Feb 22 14:25:05 2001
Scan completed successfully.
Scanning file system metadata, phase 2 ...
Scan completed successfully.
Scanning file system metadata, phase 3 ...
Scan completed successfully.
Scanning user file metadata ...
 100 % complete on Thu Feb 22 14:35:04 2001
Scan completed successfully.
Done

host1t:/gpfs1> mmlsdisk gpfs1
disk         driver   sector failure holds    holds
name         type       size   group metadata data  status        availability
------------ -------- ------ ------- -------- ----- ------------- ------------
gpfslv0      disk        512       1 yes      yes   ready         up
gpfslv1      disk        512       1 yes      yes   ready         up
gpfslv2      disk        512       1 yes      yes   ready         up
gpfslv3      disk        512       1 yes      yes   ready         up
gpfslv4      disk        512       1 yes      yes   ready         up
gpfslv5      disk        512       1 yes      yes   ready         up
gpfslv6      disk        512       1 yes      yes   ready         up
gpfslv7      disk        512       1 yes      yes   ready         up
gpfslv8      disk        512       1 yes      yes   ready         up
gpfslv9      disk        512       1 yes      yes   ready         up
gpfslv10     disk        512       1 yes      yes   ready         up
gpfslv11     disk        512       1 yes      yes   ready         up
gpfslv12     disk        512       1 yes      yes   ready         up
gpfslv13     disk        512       1 yes      yes   ready         up
gpfslv14     disk        512       1 yes      yes   ready         up
gpfslv15     disk        512       1 yes      yes   ready         up
```

7.1.7 Managing GPFS quotas

Limiting user access to the file space provided by GPFS might be required in certain instances. For this eventuality, quotas exist. Quota limit checking can be applied to groups, users, or both. Important options consist of soft limit, hard limit, and grace period. GPFS clusters require a /.rhosts file on every node.

Activating quotas

When running **mmcrfs**, the default is to not institute quotas on the GPFS file system. We would have to include the option **mmcrfs -Q yes** to activate quotas during creation of the file system or **mmchfs -Q yes** after the fact. When running **mmchfs**, we have to first issue **umount /gpfs1** on all the nodes or the change will not take affect. Example 7-20 shows the result of **mmlsfs** before quota activation. After activation, both user and group appear as flag values that can be enforced. Running the quota report generator **mmrepquota** before activation, yielded `no quota management installed`. Afterwards, a table is written, but is empty pending editing of the quota limit file via **mmedquota**.

Example 7-20 Activating quotas

```
host1t:/> mmlsfs gpfs1 -Q
flag value          description
---- -------------- -----------------------------------------------------
 -Q  none           Quotas enforced

host1t:/> mmrepquota -g -a
gpfs1: no quota management installed

host1t:/> mmchfs gpfs1 -Q yes
mmchfs: Propagating the changes to all affected nodes.
This is an asynchronous process.

host1t:/>  mmlsfs gpfs1 -Q
flag value          description
---- -------------- -----------------------------------------------------
 -Q  user;group     Quotas enforced

host1t:/> mmrepquota -u -a
*** Report for USR quotas on gpfs1
                         Block Limits
Name       type          KB      quota      limit   in_doubt    grace

| File Limits
|    files quota    limit in_doubt    grace
```

Establishing quotas

Once we turned quotas on, we designated different limits to different users. In Example 7-21, we created users red, green, blue and yellow and made directories (not required) under /gpfs1 for each of them. These users owned all the files within these directories.

A soft limit is the total of all that user's files and does not immediately impact the ability of that user to created or add to those files. That changes when the user exceeds the grace period (the time passed after initially going over the soft limit) or if the user hits the hard limit. Since the limits must be a multiple of the block size, we chose a soft limit of 256K, a hard limit of 256K and grace period of 2 days (default is 7 days) for user green. For user red, we picked a soft limit of 512K and a hard limit of 1024K, while the grace period is the same for all users. For the sake of experimentation, we left the inodes at zero, which means unlimited.

Example 7-21 Establishing quotas

```
host1t:/> mmedquota -u green
*** Edit quota limits for USR green
NOTE: block limits will be rounded up to the next multiple of the block size.
gpfs1: blocks in use: 0K, limits (soft = 0K, hard = 0K)
        inodes in use: 0, limits (soft = 0, hard = 0)

*** Edit quota limits for USR green
NOTE: block limits will be rounded up to the next multiple of the block size.
gpfs1: blocks in use: 0K, limits (soft = 256K, hard = 256K)
        inodes in use: 0, limits (soft = 0, hard = 0)

host1t:/> mmedquota -u red
*** Edit quota limits for USR red
NOTE: block limits will be rounded up to the next multiple of the block size.
gpfs1: blocks in use: 0K, limits (soft = 0K, hard = 0K)
        inodes in use: 0, limits (soft = 0, hard = 0)

*** Edit quota limits for USR red
NOTE: block limits will be rounded up to the next multiple of the block size.
gpfs1: blocks in use: 0K, limits (soft = 512K, hard = 1024K)
        inodes in use: 0, limits (soft = 0, hard = 0)

host1t:/>  mmedquota -t -u
*** Edit grace times
Time units may be: days, hours, minutes, or seconds
Grace period before enforcing soft limits for USRs:
```

```
gpfs1: block grace period: 7days, file grace period: 7days
*** Edit grace times
Time units may be: days, hours, minutes, or seconds
Grace period before enforcing soft limits for USRs:
gpfs1: block grace period: 2days, file grace period: 7days
```

Listing quotas

To view any established quota, as in Example 7-22, we ran **mmlsquota**, or **mmrepquota**, depending on the format of the output we wanted. As long as we did not exceed the quota for a user, grace shows up as none. When we exceed our quota, grace displays the number of days the user has to correct the over quota problem before the ability to write is curtailed. The output of these commands is very wide and separated near the middle by a “|” symbol. The **mmlsfs gpfs1 -Q** command shows the current options for the specified file system, the -Q line shows quotas are enfoced for both user and group.

Example 7-22 Listing quotas

```
host1t:/gpfs1/red> mmlsquota -u green
                         Block Limits                                    |
File
 Limits
Filesystem type          KB      quota      limit   in_doubt    grace
gpfs1      USR          456        256        256       -216    2days

|    files quota    limit in_doubt    grace
|       19 0        0       -9     none

host1t:/gpfs1/green> mmrepquota -u -a
*** Report for USR quotas on gpfs1
                         Block Limits                                    |
            File Limits
Name       type          KB      quota      limit   in_doubt    grace
root       USR     65954312          0          0       5120     none
red        USR          128        512       1024          0     none
green      USR         4056        256        256      -3728    2days
blue       USR           24          0          0          0     none
yellow     USR           16          0          0          0     none

|    files quota    limit in_doubt    grace
|       16 0        0       20     none
|        6 0        0        0     none
```

```
|       23 0        0       -2     none
|        2 0        0        0     none
|        2 0        0        0     none

host1t:/gpfs1/green> mmlsfs gpfs1 -Q
-Q  user;group     Quotas enforced
```

Deactivating quotas

In Example 7-23, we ran **mmquotaoff** to deactivate quotas. By specifying the file system name without modifiers, the command disabled both user and group quotas. Had we specified -u or -g, we could have turned off only user or group quotas respectively. When we ran **mmlsquota**, it still looked like quotas was running; we had to use **umount /gpfs1, mmchfs -Q no** to completely stop quotas and then **mount /gpfs1** before **mmlsquota** responded with `no quota management installed`.

Example 7-23 Deactivating quotas

```
host1t:/gpfs1/green> mmquotaoff gpfs1

host1t:/gpfs1/green> mmlsquota
                         Block Limits
Filesystem type          KB      quota      limit   in_doubt    grace gpfs1
USR     no limits

|     File Limits
|    files quota    limit in_doubt    grace

host1t:/gpfs1/green> mmchfs -Q no
host1t:/gpfs1/green> gdsh "umount /gpfs1"
host1t:/gpfs1/green> gdsh "mount /gpfs1"

host1t:/> gdsh "df -k | grep gpfs"
host1t: /dev/gpfs1      142077952  75916032   47%       67     1% /gpfs1
host2t: /dev/gpfs1      142077952  75916032   47%       67     1% /gpfs1
host3t: /dev/gpfs1      142077952  75916032   47%       67     1% /gpfs1
host4t: /dev/gpfs1      142077952  75916032   47%       67     1% /gpfs1

host1t:/> mmrepquota -u -a
gpfs1: no quota management installed

host1t:/> mmlsquota
                         Block Limits
```

```
Filesystem type          KB      quota      limit   in_doubt    grace

|     File Limits
|    files quota    limit in_doubt    grace gpfs1: no quota management
installed
```

7.2 HACMP administration

In this section we describe common maintenance tasks that the user will encounter.

7.2.1 Changing the cluster configuration

Changes to the cluster configuration are performed while the cluster services are active on some nodes are called Dynamic Automatic Reconfiguration Event (DARE).

Deleting and adding cluster nodes

In the following example, we will add and delete the node host2t from the HACMP/ES cluster. We perform all commands from host3t.

In Example 7-24, before removing host2t from the cluster nodeset, we stop the mmfs daemon and remove it from the GPFS configuration.

Example 7-24 Deleting a cluster node

```
host3t:/> rsh host2t mmshutdown
Thu Feb 22 18:17:07 EST 2001: mmshutdown: Starting force unmount of GPFS
filesystems
Thu Feb 22 18:17:12 EST 2001: mmshutdown: Shutting down GPFS daemons
Shutting down!
Thu Feb 22 18:17:12 EST 2001: mmshutdown: Finished
host3t:/> mmdelnode host3e
Verifying GPFS is stopped on all nodes ...
mmdelnode: Removing old nodeset information from the deleted nodes.
This is an asynchronous process.
mmdelnode: Propagating the changes to all affected nodes.
This is an asynchronous process.
host3t:/>
```

In Example 7-25 on page 138, we stop the HACMP/ES cluster services on host2t and wait until the cluster is stable. A dynamic reconfiguration event can only be performed when the cluster is stable.

Example 7-25 Stop cluster services

```
host3t:/> rsh host2t '/usr/es/sbin/cluster/utilities/clstop -y -N -g'
Feb 22 2001 18:23:20/usr/es/sbin/cluster/utilities/clstop: called with flags -y
-N -g

0513-044 The clstrmgrES Subsystem was requested to stop.
0513-044 The clsmuxpdES Subsystem was requested to stop.
0513-044 The clinfoES Subsystem was requested to stop.
```

We wait until the cluster is stable, as displayed by the **clstat** command, as shown in Example 7-26.

Example 7-26 Execute **clastat** command

```
host3t:/> /usr/es/sbin/cluster/clstat
```

Next, in Example 7-27, we remove host2t from the nodeset of the cluster. This will remove all cluster adapters that are configured on the node as well. Note that all following commands have to be performed on a node on which the cluster services are active; this is a requirement for DARE. We chose host3t.

Example 7-27 Remove cluster node

```
host3t:/> smitty hacmp
   Cluster Configuration
      Cluster Topology
         Configure Nodes
            Remove a Cluster Node
            (select host2t)
```

The node now has been removed from the node set in the HACMP/ES cluster configuration on host3t, and all cluster adapters that were configured on it.

In Example 7-28, we synchronize the cluster topology on host3t. This will distribute the change made locally to the cluster topology on host3t to all other nodes in the cluster. It will update the cluster configuration that is known to the cluster manager. The cluster manager daemons will run a reconfig_topology event.

Example 7-28 Synchronize cluster topology

```
host3t:\> /usr/es/sbin/cluster/utilities/cldare -t
```

The cluster verification cannot be skipped when synchronizing the cluster configuration while the cluster services are active on some nodes, since an invalid cluster configuration could cause the cluster to go into the error state.

In Example 7-29, we monitor the events that are run during the DARE.

Example 7-29 Verify successful synchronization

```
host3t:/> more /usr/es/sbin/cluster/history/cluster.0306
Feb 22  18:28:39 EVENT START: node_down host2t graceful
Feb 22  18:28:39 EVENT COMPLETED: node_down host2t graceful
Feb 22  18:28:40 EVENT START: node_down_complete host2t graceful
Feb 22  18:28:41 EVENT COMPLETED: node_down_complete host2t graceful
Feb 22  18:34:00 EVENT START: reconfig_topology_start
Feb 22  18:34:01 EVENT COMPLETED: reconfig_topology_start
Feb 22  18:34:01 EVENT START: reconfig_topology_complete
Feb 22  18:34:05 EVENT COMPLETED: reconfig_topology_complete
```

We can convince ourselves that host2t has been removed from the cluster topology by displaying the cluster networks, as in Example 7-30.

Example 7-30 Display cluster network

```
host3t:/> /usr/es/sbin/cluster/utilities/cllsnw
Network         Attribute  Node       Adapter(s)

gpfs_net        public     host1t     host1e
                             host3t     host3e
                           host4t     host4e

noname_net      public     host1t     host1t
                           host3t     host3t
                           host4t     host4t
```

In Example 7-31, we add host2t back to the nodeset of the cluster.

Example 7-31 Add node

```
host3t:/> /usr/es/sbin/cluster/utilities/clnodename -a host2t
```

Afterwards, inExample 7-32, we need to add the cluster adapters, host3t and host3e, that were previously configured.

Example 7-32 Add cluster adapters

```
host3t:/> /usr/es/sbin/cluster/utilities/claddnode -a host2t:token:\
> noname_net:public:service : : -n host2t
host3t:/> /usr/es/sbin/cluster/utilities/claddnode -a host2e:ether:\
> gpfs_net:public:service : : -n host2t
```

In Example 7-33, we synchronize the cluster topology.

Example 7-33 Synchronize cluster topology

```
host3t:/> /usr/es/sbin/cluster/utilities/cldare -t
```

Example 7-34 shows the sequence of events that is run while the reconfiguration is performed.

Example 7-34 Cluster reconfiguration events

```
host1t:/> tail -f /usr/es/sbin/cluster/history/cluster.03062001
Feb 22  19:09:38 EVENT START: reconfig_topology_start
Feb 22  19:09:38 EVENT COMPLETED: reconfig_topology_start
Feb 22  19:09:39 EVENT START: reconfig_topology_complete
Feb 22  19:09:42 EVENT COMPLETED: reconfig_topology_complete
```

In Example 7-35, we restart the cluster services on host2t.

Example 7-35 Restart cluster services

```
host3t:/> rsh host2t '/usr/es/sbin/cluster/etc/rc.cluster -boot -N -b -i'
```

Finally, we add host2t back to the GPFS nodeset and start the GPFS daemon. First, we determine that quorum exists for the node set. Otherwise, the adding of a new node may not be successful. In Example 7-36, we can obtain the number of active nodes in the GPFS nodeset from the long listing for the Group Services subsystem

Example 7-36 First group services subsystems

```
host3t:/> lssrc -ls grpsvcs
Subsystem         Group            PID     Status
 grpsvcs          grpsvcs          10204   active
4 locally-connected clients.  Their PIDs:
16582(hagsglsmd) 14028(haemd) 16018(clstrmgr) 14166(mmfsd)
HA Group Services domain information:
Domain established by node 3
Number of groups known locally: 5
                   Number of   Number of local
Group name         providers   providers/subscribers
Gpfs.set1                4           1           0
GpfsRec.set1             4           1           0
ha_em_peers              4           1           0
CLRESMGRD_111            4           1           0
CLSTRMGR_111             4           1           0
host3t:/>
```

The command shows that the groups Gpfs.set1 and GpfsRec.set1 have four providers, hence GPFS is active on all four nodes and quorum exits.

7.2.2 Changing the network configuration

The user may encounter the need to add or remove network adapters, or to change the rate at which keepalive signals are sent, or update the grace period for the detection of failures.

Adding and removing network adapters

In Section 7.2.1, “Changing the cluster configuration” on page 137, we discussed how to change the network configuration while the cluster is active. Network adapters can be added or deleted as shown Section 7.2.1. While doing so when the cluster is active, for instance to exchange a network, any new network should be added before another one is removed, to maintain redundant network connections at any moment.

Configuration of cluster network modules

In Example 7-37, for each network type (i.e., Ethernet and Token Ring), Topology Services sends keepalive signals with a certain frequency, and ignores missing responses from adapters for a certain time span before declaring the adapter has failed.

Example 7-37 Configure cluster network modules

```
host1t:/> smitty hacmp
Cluster Configuration
   Cluster Topology
      Configure Network Modules
         Change/Show a Network Module
         (we choose ether)
         (

[* Network Module Name                                  ether
  Description                                           [Ethernet Protocol]
  Parameters                                            []
  Grace Period                                          [30]
  Failure Detection Rate                                Custom
  Failure Cycle                                         [4]
  Heartbeat Rate (in tenths of a second)                [5]

  Note:  A changed value in the Heartbeat Rate
  field will be ignored if the Failure Detection
  Rate is not set to "Custom".
```

The Heartbeat Rate is the frequency keepalive signals are sent. The Grace Period is the time span keepalive signals from an adapter can be missed before Topology Services declares this adapter has failed. The network module settings are configured by network type. In this example, the settings would apply to *all* cluster adapters of type ether. A short grace period is desirable, however no false adapter failures should be generated, due to a saturated network. We tried the above settings on the network used by GPFS and did not see any errors. An adapter failure would be visible in the event history in our configuration as a network_down event.

After the network module setting has been changed, the cluster topology needs to be synchronized to update Topology Services on all nodes.

8

Developing Application Programs that use GPFS

This chapter describes concepts and provides guidelines related to the development of application programs using GPFS in a clustered environment. This chapter revisits many of the concepts introduced in Chapter 2, “More about GPFS” on page 13, but discusses them as they affect application programming performance. With this focus, numerous benchmark results are provided. Since this book is being written for an environment without PSSP, it is assumed that the programs are not parallel (i.e., MPI nor OpenMP is used); however, many of the concepts in this chapter are equally applicable to parallel programs. See *GPFS for AIX: Administration and Programming Reference* for more details.

In particular, this chapter includes:

- The relationship of the POSIX I/O API to GPFS
- The relationship of GPFS architecture and organization to application programming
- The analysis of several I/O access patterns and examples of how to improve the random I/O access pattern using the GPFS Multiple Access Range hint
- Multinode performance considerations
- Numerous benchmark results and performance monitoring

8.1 GPFS, POSIX and application program portability

This section discusses the relationship of GPFS to the POSIX I/O API and the portability of programs designed to use GPFS.

8.1.1 GPFS and the POSIX I/O API

The POSIX API[1], including its I/O routines, is a common standard that most UNIX implementations adhere to (including AIX). GPFS is designed to work with programs using this API in a simple and straight forward manner. By sticking to this standard, an application programmer can write a program without concern for the fact that they are using a GPFS file system and can expect it to work correctly on GPFS as well as any other UNIX and UNIX file system combination adhering to the POSIX standard. For example, a programmer writing code for a GPFS file system need not worry about where the disks are mounted (even though they may be mounted on multiple nodes). There are no special calls to send data to some other node or directives to open the file over several nodes. Assuming that the application program is not a parallel program (i.e., MPI or PVM is not used), the programmer can write the program in an intuitively clear and straight forward manner without concern for the distributed nature of the GPFS file system and expect the program to work correctly. But while sticking to this standard guarantees program correctness, it does not imply anything about efficiency.

A simple example

The following code segment illustrates the simple way GPFS can be used. It directly uses the familiar POSIX APIs open(), lseek(), write(), fsync() and close(). But there are no specialized system calls, no macros, nor anything else exceptional.

```
offset_t seek_offset;
int      fd;
int      k;
char     fname = “my_file”;
char     buf[16384];
int      buf_size;
. . .
fd = open(fname, O_RDWR|O_CREAT|O_TRUNC, 0777);
for (k = 0; k < 10000; k++)
{
   seek_offset = do_something(buf_size, buf);
   lseek(fd, seek_offset, SEEK_SET);
   write(fd, buf, buf_size);
```

[1] See *POSIX Programmer's Guide,* Donald Levine, O'Reilly and Associates, April 1991, ISBN 0-937175-73-0 for further details.

```
}
fsync(fd);
close(fd);
```

By defining the _LARGE_FILES flag, this code segment can access files exceeding 2 GB. See Section 8.8.2, “Notes on large files” on page 181 for further details.

While production codes are far more complex in practice, any program using the POSIX API which works correctly on a sequential file system–such as the Journaled File System (JFS)–will work correctly using GPFS, but with several added benefits. For instance, this program can safely read or write a file distributed over several disks in parallel. In other words, the program can be written using a conventional sequential style and GPFS will automatically parallelize the I/O operations. It also includes the ability to access the file from any of the several nodes where the GPFS file system is simultaneously mounted.

Note on terminology

In many RS6000 and AIX manuals, the reader encounters the term *system call*. This generally refers to the API presented by AIX to the application programmer and it includes the POSIX API calls. The authors use terms like *AIX system call* and *GPFS system call* to distinguish between API calls generally available to application programmers via AIX and more restricted ones like those associated with GPFS. If the term system call is used without qualification, assume that it means an AIX system call.

8.1.2 Application program portability

There are three perspectives regarding portability to consider.

1. Can the program work correctly on different UNIX file systems?
2. Can the program work efficiently on different UNIX file systems?
3. Can the program work efficiently on GPFS by relaxing the requirement that it work on different UNIX file systems?

The preceding example illustrates the first point that programs sticking to the POSIX standard naturally work correctly on other UNIX file systems. But while programs using the POSIX standard may work correctly on GPFS and other UNIX file systems, their performance may not be as good as desired. So when necessary, the application programmer can tune their codes to allow GPFS to more easily exploit I/O access patterns and thereby improve performance. Even this can be done most of the time when sticking to the POSIX standard. However, while this tuned code works faster using GPFS and works correctly on other UNIX and UNIX file system combinations, it may not perform as well on these other systems. While it is possible to tune some programs to work reasonably

well across multiple file systems, often subtle changes which significantly improve a program on one file system may not have the same effect on others. So regarding the second point, POSIX portability guarantees only correctness, not performance standards.

Finally, regarding the third point, GPFS provides additional AIX system calls that go beyond the POSIX standard to improve performance in some select cases. These include the GPFS hints and data shipping features. Programs which use these features can, under the right circumstances, experience a significant performance lift, but programs using these features are not generally portable to other UNIX and UNIX file system combinations.

8.1.3 More complex examples

Appendix G, “Benchmark and Example Code” on page 237 contains references to complete programs using GPFS and a source listing for a middle layer utility code used by several of these programs. They serve both as benchmark programs and as nontrivial examples of how to write programs effectively using GPFS. They are extensively commented to facilitate the reader’s understanding of GPFS programming principles. Appendix H, “Additional material” on page 259 explains how these programs can be downloaded. Also see the *GPFS for AIX: Administration and Programming Reference* for programming guidelines.

8.2 Benchmark programs, configuration and metrics

This chapter cites a number of results based upon the execution of various benchmarks. The benchmark programs are described in Appendix G, “Benchmark and Example Code” and can be downloaded as described in Appendix H, “Additional material” on page 259. These programs have detailed user comments in their comment headers (reminiscent of man pages) to assist the reader in executing them. Because benchmark results are dependent upon the particular configuration they are executed on, Table 8-1 summarizes the key parameters of the configuration used to produce them. Unless stated otherwise, the reader should assume these apply to any benchmark result given in this chapter.

Table 8-1 Benchmark system configuration

Parameter	Value
Node model	7025-F50
Number of nodes	4
Number of CPUs per node	4

Parameter	Value
Memory	1.5 GB
GPFS block size	256 KB
Disk model	7133-D40 SSA
Number of disks	16 disks on 1 loop
Logical volumes	1 disk per logical volume
Configured file system size	135 GB
GPFS network	100 MB ether
Maximum number of benchmark tasks per node	1

In addition to the configuration parameters, it is also important to understand how the results are measured. These programs use rtc() from libxlf90 to measure time. This is a low overhead timer compared to the more familiar gettimeofday() timer and can make relatively fine grained measurements. The exact granularity of rtc() is dependent upon the model of the CPU being used (but is generally, at most, a couple microseconds). The I/O rates reported in the tables are measured in MB/s reported to three significant figures (though the raw values generally have more). This rate is determined by counting the total data processed by the program divided by the total time. The timer is started just after the file has been opened till just after it is closed; it does not measure the overhead of processing user parameters and other such things. fsync() is called just before closing the file and getting the final time. The programs can simulate either an I/O intensive profile (e.g., 99%) or a balanced CPU and I/O intensive profile. Appendix G, “Benchmark and Example Code” on page 237 provides examples of the output listings from the benchmark programs.

8.3 GPFS architecture and application programming

As pointed out earlier, GPFS is simple to use by application programmers. Yet it is profoundly subtle with many opportunities available to improve application program performance. This section describes aspects of the GPFS architecture and its organization that can be exploited by the application programmer to improve I/O performance.

8.3.1 Blocks and striping

When writing an application program that uses file I/O, the programmer reads or writes a *record*. In a UNIX environment, a record is a set of bytes whose size is arbitrary. For example, in one operation a program could write a 262,144 byte record and in the next operation read a 42 byte record. While this arbitrary record size provides great flexibility for the read() and write() system calls, it has a significant impact upon the performance of I/O operations in GPFS (and most other file systems).

By contrast, a GPFS *block* has a fixed size and it is set when the file system is created, and it is not easily changed. Moreover, a block is generally, but not always, the basic granularity of a GPFS I/O operation. In other words, regardless of the size of the record specified by a read() or write() system call, the corresponding GPFS operation is performed in units of blocks, the number of blocks determined automatically by GPFS to accommodate the size of the record.

However, there are exceptions to this rule. If the application program specifies a record that is smaller than a block or that touches a fragment that covers less than a block, and no other portion of the block is accessed during its life in the GPFS pagepool, then only the corresponding subblocks are transferred between disk and the pagepool. This can save overhead by not having to wait for the disk to spin under the head as long. But this is only relative; utilizing full blocks is generally more efficient. If a subblock is accessed, all other parts of the transaction overhead still occur, but are now amortized over a small unit of transfer. Examples of where subblocks are transferred include strided or random access I/O operations.

When records do not align with block boundaries

A common difficulty when writing an application program to be used with a GPFS file system is to align records with block boundaries. Figure 8-1 on page 149 illustrates this with a specific example. Assume the GPFS block size is 256 KB, the record is 288 KB and the seek offset is 1342169088 which is 1310712K (represented by “A” in Figure 8-1). The seek offset closest to A that is aligned to a block boundary is A+8K or 1310720K. Therefore this record spans three blocks. Suppose the I/O operation is a *read* and these blocks are not already in the pagepool (i.e., GPFS cache). Then GPFS will read the complete block in the middle and one subblock before and another after it from three separate disks at the same time.

Now, suppose that the seek offset is 1310742K. Then this record would span only two blocks. Either case is relatively inefficient. These operations can be significantly improved by reading a 256KB record with a seek offset of 1310720. Then the record would be block aligned and only one block would be accessed. But aligning records on block boundaries like this is not always possible. A similar problem occurs when a program *writes* a record that does not align properly with a block boundary.

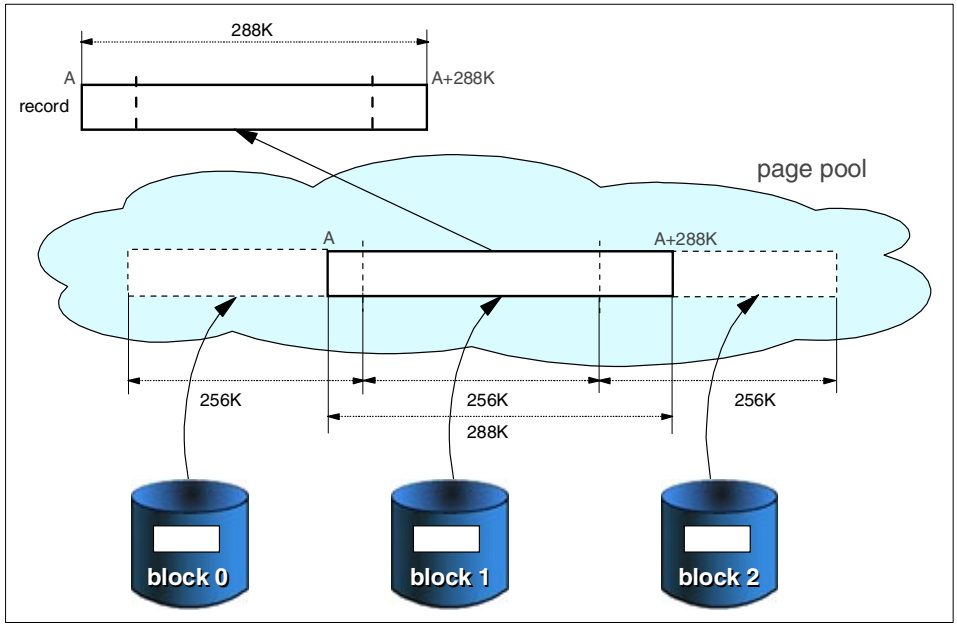


Figure 8-1 Reading a record that does not align with the GPFS block boundary

Table 8-2 on page 150 provides benchmark I/O rates corresponding to Figure 8-1. In order to eliminate other cache optimizations, this benchmark writes a 5 GB file using a random access pattern with the given record sizes. As can be seen, the 256 KB record case significantly out performs the 288 KB record case. However, if the program can be redesigned to use a record size that aligns with the block boundary, then significant savings could be realized. For example, suppose the natural record size is indeed 288 KB and the records are accessed in a random pattern. Then the program could be redesigned to read 512 KB block aligned records which contain the 288 KB of data needed. Using the same benchmark program, the raw I/O rate is now 58.3 MB/s. However, since the

access pattern is random, the file must be read twice over to access all of the data. This means that it takes twice as long to execute, thus reducing the effective I/O rate to 29.2 MB/s. Even so, 29.2 MB/s is significantly better than the raw rate of 12.8 MB/s when reading 288 KB records.

Table 8-2 Record/block mis-alignment

Record size	256 KB	288 KB
I/O rate	52.6 MB/s	12.8 MB/s

When records can be striped

Table 8-3 illustrates that a related problem occurs when a record is smaller than a block. Suppose the record is only 16 KB and the block size is 256 KB. Regardless of the seek offset, one block is read or written and the utilization is only 6.25% for this transaction. Obviously, larger records improve the utilization. The ideal is to have a record whose size is a multiple of the block size.

On the other hand, when records are significantly larger than a block, the parallel operation of GPFS striping pays significant dividends. Suppose the block size is 256 KB, and the record size 1024 KB and it is properly aligned with a block boundary. Then, ideally, four blocks are accessed simultaneously in the same amount of time it takes to read one block (due to parallel overhead, its not quite this good, but it is significantly improved). In general, designing an application program to use large records that are a multiple of the block size leads to optimum I/O performance for GPFS.

Table 8-3 Matching record and block sizes

Record size	16 KB	256 KB	1024 KB
I/O rate	1.39 MB/s	13.20 MB/s	29.70 MB/s

8.3.2 Token management

Suppose job 1 is writing to file X. Now suppose job 2 starts and also wants to write to file X. This situation requires careful coordination to prevent corrupting file X. For instance, the file system could lock file X as soon as job 1 starts writing to it and keeps it locked until job 1 is complete. In the meantime, job 2 remains idle. This technique is called *file locking*, and while it is effective, it can be inefficient. Another alternative would be to lock only that portion of file X that job 1 is writing to and allow job 2 to write concurrently to another section of the file. If job 2 does try to write to the same section job 1 is, then job 2 would have to wait. This technique is called *byte range locking* and is more efficient than file locking as multiple tasks or threads can access the file simultaneously. Now suppose these jobs are running on different nodes. This is a routine situation and it is handled by the *token manager* in GPFS.

Consider Figure 8-2. Let file X be the 8 GB file in this figure and without loss of generality let job 1 run on node 1. Now, consider jobs 3 and 4 in particular. Perhaps job 3 is writing in the byte range 4 G to 6.5 G and job 4 desires to write in the byte range 6 G to 8 G. That poses a potential write conflict. In this case, before job 4 can write to the conflicted area, the GPFS daemon must acquire a token from the token manager.

Subblocks also play a role in the token manager as the fundamental granularity for byte range locking is the subblock.

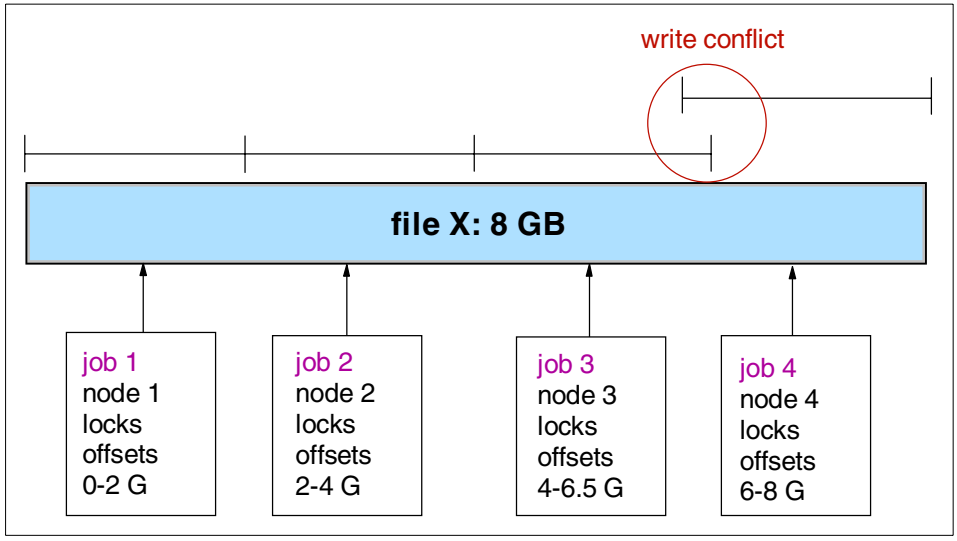


Figure 8-2 Byte range locking

GPFS is designed to support parallel programming models which simultaneously modify, or modify and read, common files, but it requires careful design of the application program. Since parallel programming is beyond the scope of this book and the token manager's activities are more critical to the parallel programming model, we will not further elaborate upon this point. However, the reader must be cautious to not allow programs to simultaneously modify, or modify and read, a common file without proper design as GPFS does not lock files being modified. If a sequential program is allowed to modify a file while some other program is accessing it, GPFS will not generate a warning, as it is designed to support such activities with properly designed programs.

8.3.3 The read and write I/O operations

The GPFS I/O operations, taking the largest of bulk of time and therefore having the most significant impact upon I/O performance, are the read and write I/O operations. The complexity of these I/O operations for sequential programs can be largely understood in terms of locality of reference or more simply, *locality*.[2] Without going into detail, good locality leads to the situation where a given record is accessed multiple times after first being placed in cache, or where an access pattern can be predicted, and the relevant records placed in cache asynchronously prior to their first access. The cache in this case is the GPFS pagepool (see Section 2.4, “Memory utilization” on page 23). This leads to two levels of complexity for understanding the read and write I/O operations.

1. The record is in the pagepool
2. The record must be accessed directly on disk

These I/O operations begin with an AIX system call (i.e., read() or write()) that leads to a GPFS call in the GPFS kernel extension. The AIX system call presents the I/O request to GPFS on behalf of the user. It performs routine tasks like collecting application program parameters, initializing GPFS structures, collecting error return codes upon failure and so forth. But it does not buffer the data; the application program record is copied straight to the pagepool when writing, or the record is retrieved directly from the pagepool to an application program supplied buffer when reading. If the data being accessed is in the pagepool, GPFS kernel extensions resolve the request without resorting to daemons, but if the data is not there, then GPFS daemons are dispatched to acquire the data.

The read I/O operation

Consider the read I/O operation from its two levels of complexity.

1. The record is contained in the pagepool

   This situation results when data from an early operation is being used again or when the access pattern had been predicted and the data prefetched. In this case, no GPFS daemons are involved and the data being read is transferred from the pagepool to a user buffer. This operation is very efficient.

2. The record is *not* contained in the pagepool

   In this case, a request is sent to a daemon which allocates a buffer in the cache, locks in the byte range (to prevent the token from being stolen) and initiates the I/O operation to the device holding the data. The process initiating the request blocks until notified by the daemon that the data is available.

[2] Since the activities of the token manager are greatly subdued (but not eliminated) for sequential application programs, its role for read and write I/O operations is not considered here. See *GPFS for AIX: Concepts, Planning, and Installation Guide* for further details.

At the completion of the read operation a determination is made as to whether additional data should be prefetched. This is called *read-ahead*. A determination is made as to how much data should be prefetched based upon the program's I/O execution profile. A daemon asynchronously goes about prefetching the data as necessary. Sequential and strided access patterns effectively exploit read-ahead (see Section 8.4.1, "Tables of benchmark results" on page 155). Hints can also be supplied when the pattern is random to assist read-ahead (see Section 8.5, "Hints: Improving the random I/O access pattern" on page 160).

The write I/O operation

When an application program running under AIX writes to a GPFS file using the write() system call, a record is copied from the application program's buffer to the GPFS cache (and no other copies are made). Generally, it is scheduled for *flushing* at some later time to increase performance. Flushing is the process of physically placing the data that is in cache on the disk. But flushing does not necessarily mean that the data has been removed from cache. This process of scheduling the flushing operation sometime later is called *write-behind*.

A block is scheduled to be flushed when one or more of the following situations occur:

- The application specifies a synchronous write
- The application program calls a sync() system call
- The last byte of a block on sequential write has been written
- The cache is full and room is needed for a new block
- Another process is writing to the same file block

The last item guarantees cache coherence. A daemon actually does the flushing. Unless specified otherwise (as in the first case), the flush is done *asynchronously* (i.e., independently of, and concurrently with, the application program threads).

As with the read case, consider the write I/O operation from its two levels of complexity.

1. The record is contained in the pagepool

   When we say that the record is contained in the pagepool, we mean that the GPFS blocks that will contain the record are in the pagepool. In this case, the GPFS kernel extension copies the record from the buffer provided by the application program directly to the pagepool. Unless this is a synchronous write, control is then returned to the application program and if one of the situations occur forcing a flush, it is scheduled and processed asynchronously by a daemon.

2. The record is *not* contained in the pagepool

If the blocks that will contain the record are not present in the pagepool, the GPFS kernel extension suspends the application program and dispatches a thread to allocate the blocks before writing the record. There are three options to consider.

- A partial block is being written

  If a disjoint part of the block has already been written to, then the block is first copied from disk to cache and the portion of disk now being written to is overlaid by the current record. When it is flushed, it replaces the old block on disk.

- A complete block is being written

  There is no reason to copy a block from disk to cache. An empty block in cache is allocated and the record is written to it. When it is flushed, it replaces the old block on disk.

- A new block is being written

  The daemon allocates an empty block in cache and assigns it to a block in the file. When it is flushed, it is copied to a new location on the disk.

Design strategies that exploit locality

From this analysis, it can be seen that application program designs that exploit cache locality are more efficient than those that do not. For example, designing an application program that references records multiple times in quick succession, or that allows prefetching, is far more efficient than those that don't.

8.4 Analysis of I/O access patterns

This section analyzes *I/O access patterns*. We use this term to refer to the order in which records are accessed and the topology of the records (e.g., size, block alignment). It is the I/O access pattern that determines how efficiently GPFS can execute I/O operations. Where possible, GPFS automatically utilizes caching, striping and record/boundary alignment with respect to the I/O access pattern to improve I/O performance. By designing an application program to use an I/O access pattern that can easily be exploited, overall program performance can be enhanced. While designing a program to use one of these efficient access patterns may not be the natural way to do it, if it is possible, the benefits can make the effort worth while. For example, we know of one case where a production program was improved 10X by redesigning it to use a sequential access pattern with large records, rather than a random access pattern with small records.

8.4.1 Tables of benchmark results

It is not possible to analyze every conceivable I/O access pattern. However, a range of relatively common patterns with varying degrees of efficiency are discussed, which illustrate the principles discussed earlier in this chapter. But before actually discussing these patterns, Table 8-4 summarizes the results of benchmark tests measuring the I/O rates of each access pattern. They are based on the programs referenced in Appendix G, “Benchmark and Example Code” on page 237. These are sequential programs and they were executed on idle systems (i.e., they were the only jobs running on any of the four nodes in the system). Section 8.2, “Benchmark programs, configuration and metrics” on page 146 summarizes the configuration parameters used for these benchmarks. In addition, the following information pertains to this table:

- Three record sizes were used:
  - L = 1024 KB (large)
  - M = 256 KB (medium) which is the block size for this configuration
  - S = 16 KB (small)
- Records are block aligned and have a fixed size
- The file size for each job is always 5 GB

The programs represented in the columns designated “I/O only” perform no CPU tasks beyond the immediate needs of doing the I/O operation; the time devoted to CPU activity is a fraction of a percent. The programs represented in the columns designated “I/O and CPU” are balanced between CPU and I/O activity; the time devoted to CPU activity is somewhat variable, depending on the pattern, but typically ranges between 50% to 60%.

Table 8-4 Benchmark results for I/O access pattern tests

Record size-Pattern	Write I/O Only	Read I/O Only	Write I/O and CPU	Read I/O and CPU
L-Sequential	67.00 MB/s	75.90 MB/s	32.70 MB/s	30.80 MB/s
M-Sequential	64.00 MB/s	73.2 MB/s	32.60 MB/s	30.60 MB/s
S-Sequential	66.80 MB/s	69.80 MB/s	34.9 MB/s	35.90 MB/s
L-Strided	65.90 MB/s	75.20 MB/s	32.80 MB/s	30.70 MB/s
M-Strided	64.80 MB/s	72.80 MB/s	32.00 MB/s	30.50 MB/s
S-Strided	10.70 MB/s	19.90 MB/s	10.70 MB/s	20.60 MB/s
L-Random	64.40 MB/s	29.80 MB/s	32.40 MB/s	19.00 MB/s
M-Random	52.60 MB/s	13.20 MB/s	21.50 MB/s	10.60 MB/s

Record size-Pattern	Write I/O Only	Read I/O Only	Write I/O and CPU	Read I/O and CPU
S-Random	6.23 MB/s	1.39 MB/s	5.64 MB/s	1.36 MB/s

We recommended that you execute these benchmarks on your local systems since the results of a benchmark are heavily dependent upon the system (and its configuration) on which they are executed. Therefore, if the system configuration is different, the values will also be different. Moreover, even if the systems are identical, the results may not be identical, as there is some variance (as much as 10% or more) associated with these figures. But the trends should be similar.

Each of the I/O access patterns listed in Table 8-4 are now discussed. They are presented in an approximate hierarchy from the most effective to the least effective.

8.4.2 Sequential I/O access patterns

A sequential I/O access pattern involves accessing records one after the other starting at the beginning of the file and going to the end. This is by far the most efficient I/O access pattern for GPFS.

When the records are large (e.g., 1024 KB) relative to the block size (e.g., 256 KB), GPFS is able to stripe these records across several disks improving performance using implicit parallelism; each record accessed can spin multiple disks at the same time. Moreover, GPFS's read-ahead and write-behind algorithms detect the sequential locality and schedule transfers between disk and cache optimally. This can be seen by comparing the results of the large record sequential patterns with the large record random patterns in Table 8-4. In the large record random case, access is improved by striping, but the read-ahead and (to a lesser extent) write-behind are not as effective and the performance is significantly less than in the corresponding sequential case.

But even if the records are medium (e.g., 256 KB) or small (e.g., 16 KB), the read-ahead and write-behind algorithms detect the sequential locality, group these records into multiple blocks, and stripe them as in the large record case. These observations are consistent with the figures in Table 8-4 on page 155. The differences between the large, medium and small record reads and writes are statistically insignificant.

8.4.3 Strided I/O access patterns

When discussing strided I/O access pattern, there is a *stride factor* of N associated with it. A stride of N means that a record is accessed, the next N-1 records are skipped, a record is accessed, the next N-1 records are skipped, and so on. Once the program gets to the end of the file, it goes back to the beginning and starts with the next un-accessed record and repeats the strided pattern again shifted by one record. The relative efficiency of this method is dependent on the size of the record. But even if the records are smaller, GPFS accelerates this access pattern.

First consider the case when the records are large (e.g., 1024 KB) relative to the block size (e.g., 256 KB). Large blocks like this are striped and efficiency is very good due to the implicit parallelism associated with striping. Its performance is statistically identical to large record sequential.

Next consider the case when the records are medium sized (i.e., the same as the block size or 256 KB). In earlier versions of GPFS, the performance would have been identical to a random I/O access pattern, but in the later versions of GPFS, it can detect this strided pattern; when it does, it either pre fetches blocks asynchronously when it is reading a file, or it optimally schedules the records to be written to disk after being placed in the pagepool using write-behind. Thus, the medium record strided pattern performs statistically the same as large record strided and the sequential patterns[3].

A significant performance degradation is observed when the record size is small (e.g., 16 KB) relative to the block size (e.g., 256 KB), especially of reads. But the performance is still better than a random I/O access pattern for the same block size. The reason is that GPFS can detect the strided pattern and read-ahead or write-behind accordingly, but since it only uses a small portion of the block, the overhead of reading an entire block is amortized over the small record. It is like using a large pipe to transport a trickle of water. When measuring I/O rates based on the amount of data accessed by the application program, the rate for the small record program is proportional to the rate of the same program using records equal to the block size where the constant of proportionality is the fraction of the record size to the block size. This assumes that stride N is such that two or more records come from one block. This relation can be expressed more clearly mathematically:

- Let Q_L be a record whose size is equal to one GPFS block.
- Let Q_S be a small record size with $Q_S < Q_L$.

[3] This statistical identical performance to sequential access patterns of the medium and large record strided writes is surprising to the authors. On SP systems using VSD servers, the write performance is relatively good but is only 60% of sequential access patterns.

- Let n be the number of records accessed per block with n > 1. n is inversely proportional to the stride N.
- Let $F = Q_S / Q_R$.
- Let R_L be the I/O rate using the large records Q_L.
- Let R_S be the I/O rate using the small records Q_S.

Then we conclude:

- $R_S = k_1 * n * F * R_L + k_2$ where k_1 and k_2 are constants.

or more simply, the rate $R_S = O(n)$. In other words, when the record size is less than the block size and n > 1, the I/O rate varies linearly with the number of records accessed per block. Since n = O(1/N), we can also say that the I/O rate is inversely proportional to the stride. When n = 1, the constants k_1 and k_2 change because GPFS now reads a subblock.

Table 8-5 shows this relationship. In this example, the benchmark programs write or read a 1 GB file with a record size of 16 KB. The block size is 256 KB. Thus there is 16 records per block. Note that each time the stride doubles, the rate is cut in half, until the stride = 16 and the constants k_1 and k_2 change. When stride = 1, the I/O access pattern becomes small record sequential and striping becomes operational. The rates in this case are 66.2 MB/s for the write and 75.9 MB/s for the read.

Table 8-5 Relationship between stride and I/O rate for small record strided

Stride (N)	Records per block (n)	Write rate (R_S)	Read rate (R_S)
2	8	24.30 MB/s	40.70 MB/s
4	4	10.60 MB/s	19.70 MB/s
8	2	5.03 MB/s	9.56 MB/s
16	1	9.73 MB/s	22.10 MB/s

Summarizing, large and medium record strided I/O access patterns perform at the same rate as sequential I/O access patterns; small record strided I/O access do not perform as well, but perform better than a random I/O access pattern for blocks of the same size.

8.4.4 Random I/O access patterns

Random I/O access patterns are problematic, but for many applications this method cannot be avoided. The problem is that GPFS cannot detect a pattern. Therefore impact of the read-ahead and write-behind algorithms is significantly reduced.

First, consider the case when the records are large (e.g., 1024 KB) relative to the block size (e.g., 256 KB). Here GPFS can only exploit the implicit parallelism from striping a large record. Because no pattern can be detected, read-ahead cannot be exploited at with any statistically significant impact. In the write case, the scheduling of transfers from the pagepool to disk using write-behind are much more efficient and perform at a rate nearly the same as the sequential patterns.

The other cases are harder to analyze. Consider the read case when the records are medium (i.e., the same as the block size or 256 KB) and small (e.g., 16 KB). In the medium case, the read performance drops significantly as there is neither a pattern that can be detected and exploited by the read-ahead algorithm, nor is there any opportunity for implicit parallelism associated striping. When the records are small, only the number of subblocks needed to fill the request are read (e.g., two, 8 KB subblocks). The overhead of transfer from disk to the pagepool must be amortized over a smaller unit of transfer and performance suffers relative to the medium record case.

The analysis of the write case for the small and medium random I/O access patterns is similar to the read case when the two write cases are compared to themselves; medium record random is faster than small record random as the small case must amortize the overhead of transfer over a small unit of transfer. But what is surprising to the authors is that writes are from 2X to 4X faster than the reads. Testing parallel versions of these benchmarks on RS6000 systems using an SP2 switch and VSD servers yields just the opposite.

Incidentally, a similar mathematical relationship that exists for the small record strided I/O access pattern also exists for the small record random I/O access pattern.

8.5 Hints: Improving the random I/O access pattern

As can be seen from examining Table 8-4 on page 155, random I/O access patterns do not perform as well as sequential or strided I/O access patterns. Thus it is well worth the effort to redesign an algorithm to use one of the more efficient patterns discussed in Section 8.4, “Analysis of I/O access patterns” on page 154. However, GPFS provides a mechanism to improve random I/O access patterns when they cannot be avoided.

The problem with a random I/O access pattern is that GPFS cannot predict which records will be accessed next. This renders write-behind and prefetching in-effective. To counteract this shortcoming, GPFS provides mechanisms by which the application program can identify a list of blocks (not records) to be accessed prior to their first use. This gives GPFS the opportunity to asynchronously perform prefetching and write-behind. These mechanisms are generically called *hints*.

There are two types of hints: the *GPFS access range* hint, which specifies a single range in which future accesses will occur, and the *GPFS multiple access range* hint which is finer grained.

The API for GPFS hints is not POSIX compliant (and therefore not portable) and it can be quite cumbersome, since it was designed for the needs of MPI-IO rather than application programmers. However, a simple API that is more naturally attuned to the needs of an application program can be designed as a middle layer utility, thus allowing programmers to use some of the advanced features of GPFS without having to resort to MPI.

To this end, this section discusses the implementation and use of such a middle layer utility based upon the GPFS multiple access range hint (MAR hint). In particular, it describes:

- Selected key aspects of the MAR hint API needed by this middle layer utility
- A generic middle layer utility, called GMGH, based upon the MAR hint API to provide an easier to use hints API for application programs
- Benchmarks comparing random I/O access patterns using and not using hints

For the remainder of this section, assume the term hints is synonymous with MAR hints.

8.5.1 The GPFS Multiple Access Range hints API

In order to understand how GMGH works, it is necessary to understand the basic MAR hint API (both its syntax and associated semantics). This section does not explain every nuance and feature of this API, nor even everything used by GMGH, but it does explain in broad relief the key features of this API so that the reader can intelligently use GMGH and modify it to meet specific needs. Greater details regarding the system calls and structures associated with the MAR API can be found in *GPFS for AIX: Administration and Programming Reference* . The working code for GMGH which uses the MAR API serves as a complete example and can be found in Appendix G, “Benchmark and Example Code” on page 237.

To begin with, there are five principles guiding the semantics of the MAR hint API:

1. Hints are *suggestions* to GPFS; they are not guaranteed to be acted on.
2. Hints are issued prior to accessing a record with a read() or write() system call. For reads and writes, this allows blocks corresponding to the record to be placed in the pagepool, as necessary, asynchronously by GPFS daemon threads prior to their first use. It also assists writes with their write-behind scheduling.
3. Since a GPFS I/O operation never exceeds a block (remember, a large record is divided into multiple blocks), hints are issued for each block corresponding to the record.
4. The number of outstanding hints is limited by the system configuration. Hints are issued until no more are accepted. Hints are released for blocks that have been accessed and then new ones are issued.
5. Operationally, hints can be issued and released after each read() or write() system call.

Each component of the MAR hints API is now discussed with several examples included to illustrate their use. The examples are designed to work with large files (i.e., files that exceed 2 GB).

gpfs_fcntl()

The only system call included in the MAR hints API is gpfs_fcntl(). This is a general purpose system call which invokes different functions based upon the structure fields in the parameters being passed to it. It is similar in spirit to the familiar ioctl() system call. The syntax of this system call is:

```
#include <gpfs_fcntl.h>
int gpfs_fcntl(int fd, void *fcp)
```

The parameters are defined as:

- `fd`

  The file descriptor for the open file to which the hints are being applied

- `fcp`

  This is a pointer to a nested structure designed by the application programmer containing other GPFS defined structures; the particular structures included determine the functions to be invoked and contain the relevant parameters.

The return value is zero if gpfs_fcntl() is successful, otherwise it is -1. In the latter case, the global variable errno is set to record the specific error.

Note that the application program using this function must be linked with libgpfs.a (specified by the compiler -lgpfs).

There are three GPFS defined structures used by GMGH that can be contained in the structure pointed to by fcp. They are:

- gpfsFcntlHeader_t
- gpfsCancelHints_t
- gpfsMultipleAccessRange_t

gpfsFcntlHeader_t

This structure is used by gpfs_fcntl() to specify the version and the size of the operand (i.e., the size of the structure pointed to by fcp) being passed to it. The fields used by GMGH are:

```
typedef struct
{
   int totalLength;    /* size of the operand */
   int fcntlVersion;   /* the particular function to be issued */
   int fcntlReserved;  /* always set to 0 */
   int errorOffset;    /* not used in GMGH */
} gpfsFcntlHeader_t;
```

fcntlVersion is always set to GPFS_FCNTL_CURRENT_VERSION, whose value is set in gpfs_fcntl.h. The operand pointed to by fcp is a structure whose size and fields vary with the particular function being launched by gpfs_fcntl().

gpfsCancelHints_t

This is one of the structures used by gpfs_fcntl() to determine which GPFS functions are being invoked. Strictly speaking, the function associated with this structure is not a hint, but a *directive*; as a directive, it is not just good advice given to GPFS, but an action that must be executed.

The function associated with this structure is to remove any hints that have been issued against the open file. It restores the hint status to what it was when the file was first opened, but it does not alter the status of the GPFS cache.

The syntax of this structure is:

```
typedef struct
{
  int  structLen;
  int  structType;
} gpfsClearFileCache_t;
```

structlen specifies the size of the gpfsClearFileCache_t structure. structType specifies the function associated with the call to gpfs_fcntl(); its value is GPFS_CLEAR_FILE_CACHE and is set in gpfs_fcntl.h.

The following code segment is a simple example illustrating how gpfs_fcntl() is called and how the structures are aggregated to specify its function and form its parameters.

```
struct
{
   gpfsFcntlHeader_t hdr;
   gpfsCancelHints_t cancel;
} chint;

chint.hdr.totalLength   = sizeof(chint);
chint.hdr.fcntlVersion  = GPFS_FCNTL_CURRENT_VERSION;
chint.hdr.fcntlReserved = 0;

chint.cancel.structLen  = sizeof(gpfsCancelHints_t);
chint.cancel.structType = GPFS_CANCEL_HINTS;

if (gpfs_fcntl(fd, &chint) < 0)
{
   printf("*** ERROR *** gpfs_fcntl error:  errno = %d\n", errno);
   return -1;
}
```

gpfsMultipleAccessRange_t

This is another one of the structures used by gpfs_fcntl() to determine which GPFS functions are being invoked. It is a nested structure used to present a range of blocks to the MAR hint mechanism; blocks can be issued as a hint and/or they can be released after having been accessed.

The structure's syntax is:

```
typedef struct
{
```

```
  int               structlen;
  int               structType;
  int               accRangeCnt;
  int               relRangeCnt;
  gpfsRangeArray_t  accRangeArray[GPFS_MAX_RANGE_COUNT];
  gpfsRangeArray_t  relRangeArray[GPFS_MAX_RANGE_COUNT];
} gpfsMultipleAccessRange_t;
```

GPFS_MAX_RANGE_COUNT specifies the maximum number of blocks that can be issued as a hint and released in a single call to gpfs_fcntl(). Its value is specified in gpfs_fcntl.h. structType specifies the function associated with the call to gpfs_fcntl(); its value is GPFS_MULTIPLE_ACCESS_RANGE, since this is the MAR hint function and is also set in gpfs_fcntl.h. structlen specifies the size of the gpfsMultipleAccessRange_t structure. accRangeArray and relRangeArray are used to issue and release hints in connection with accRangeCnt and relRangeCnt.

By listing blocks in accRangeArray, they are issued as hints. As an input parameter, accRangeCnt specifies the number of entries in accRangeArray starting from the beginning of the array. After returning from gpfs_fcntl(), accRangeCnt specifies the number of hints actually accepted; it's always the first n entries in accRangeArray. If some of the hints are not accepted, they must be issued again in the next call to gpfs_fcntl().

By listing blocks in relRangeArray, they are released; relRangeCnt specifies the number of entries in relRangeArray starting from the beginning of the array. Blocks that have been issued and accepted as a hint must be released after they have been accessed (via the read() or write() system calls); if this is not done, hints will no longer be accepted. Blocks listed in relRangeArray are always accepted for release, but if these blocks do not correspond to blocks that have been issued earlier, no error occurs.

Each block in the range being hinted is described by the structure:

```
typedef struct
{
  offset_t  blockNumber;  /* data block number to access */
  int       start;        /* displacement to 1st byte in block */
  int       length;       /* number of bytes in block being accessed */
  int       isWrite;      /* 0 - read, 1 - write */
  char      padding[4];
} gpfsMultipleAccessRange_t;
```

Consider blockNumber. A file can be viewed logically as a linear collection of blocks and each block has a seek offset indexing its first byte. blockNumber is then set by doing an integer divide of this seek offset by the GPFS block size. The GPFS block size can be found as follows:

```
struct stat64 sbuf;
fstat64(fd, &sbuf);
gpfs_blksize = sbuf.st_blksize;
```

start is the displacement (relative to the block) to the first byte in the block to be accessed by the application program (for block aligned records, start is 0). length specifies the number of bytes being accessed starting from start. If the block is being written, isWrite is 0; if its being read, isWrite is 1.

Application programs think in terms of records, whereas the MAR hint API thinks in blocks. Therefore it is necessary to correlate records to the blocks they span. The following code segments illustrate one way to issue the hints corresponding to the application program records. They come from the function gmgh_issue_hint() listed in Appendix G, “Benchmark and Example Code” on page 237. References in the following discussion to actions taken elsewhere in this code (e.g., generating the hint vector and block list) are highlighted in the appendix by call-out boxes. To improve the readability of this example, error checking, debugging code segments, and comments made redundant by embedded text have been removed; it is otherwise a complete and correct function.

This first code segment is the function header.

```
int gmgh_issue_hint
(
   gmgh *p,
   int  nth
)
```

GMGH is a structure containing various parameters and lists. There are two important lists in GMGH. p->hint is a vector whose entries are pointers to structures describing each record the application program has submitted for hinting by calling gmgh_post_hint(). There can be a maximum of 128 entries in this vector. Once exhausted, the application program submits another batch of 128 entries, and so on. The number 128 is arbitrary and can be set to meet application program needs. Corresponding to this vector is the block list p->blklst; its entries are pointers to structures describing the blocks corresponding to the records in p->hint. When the application program posts its records as hints (i.e., enters them into p->hint), another function, gmgh_gen_blk(), creates corresponding entries in p->blklst. gmgh_gen_blk() determines how many blocks each record in p->blklst spans and creates one entry for each block. Since the record size is variable in this example, a maximum block size must be declared; this is used to set p->nbleh which is the maximum number of blocks per record. So if the GPFS block size is 256 KB, and the maximum record size is set to 1024 KB, then there will be four times as many entries in p->blklst as there are in p->hint.

nth specifies the current record being issued as a hint; nth is always less than p->nhve, which is the number of entries in the hint vector.

The following code segment simply defines some local variables used in the code segments that follow. Embedded comments briefly describe them, but detailed analysis of their use is explained further below.

```
{
   int    rem;                    /* remember a value */
   int    accDone, relDone;       /* while loop conditions */
   int    nhntacc;                /* number of hints accepted */
   int    rbx;                    /* block list index for released blocks */
   int    ibx;                    /* block list index for issued blocks */
```

ghint is the parameter that is passed to the gpfs_fcntl() system call. It is being intialized here.

```
   struct
   {
      gpfsFcntlHeader_t hdr;
      gpfsMultipleAccessRange_t marh;
   } ghint;

   ghint.marh.structLen  = sizeof(ghint.marh);
   ghint.marh.structType = GPFS_MULTIPLE_ACCESS_RANGE;
```

Some more routine initialization comes next. p->nbleh is the maximum number of block entries per hint; thus rbx references the next set of blocks to be hinted. relDone and accDone are Boolean values used to determine when we are done releasing and issuing hints. “acc” in accDone indicates acceptance of hints; if accDone = TRUE, we are done accepting hints, so no more can be issued.

```
   rbx = nth * p->nbleh;            /* first block in block list to release */

   if (nth == -1) relDone = TRUE;  /* no hints to release */
   else           relDone = FALSE;

   accDone = FALSE;
```

This is where we get down to business. Blocks associated with the last accessed record are released, and we continue issuing new hints (i.e., one hint per block) until no more are accepted.

```
   while(!accDone || !relDone)
   {
```

This following for loop prepares the relRangeArray structures for release of issued hint blocks. It does this by extracting the block information from p->blklst starting with p->lstblkreleased. Notice that relDone is set to TRUE in an if statement. If the record size is very large relative to the GPFS block size, then

the number of blocks corresponding to the record size may exceed GPFS_MAX_RANGE_COUNT. Since this is the maximum number of blocks that can be processed in one call to gpfs_fcntl(), then it may be necessary to cycle through the outer while loop to release the remaining blocks, even if no more hints are being accepted.

```
for (k = 0; k < GPFS_MAX_RANGE_COUNT; k++)
{
   if (p->hint[nth].lstblkreleased >= p->hint[nth].nblkstouched)
   {
      relDone = TRUE;
      break;
   }

   ghint.marh.relRangeArray[k].blockNumber = p->blklst[rbx].blknum;
   ghint.marh.relRangeArray[k].start       = p->blklst[rbx].blkoff;
   ghint.marh.relRangeArray[k].length      = p->blklst[rbx].blklen;
   ghint.marh.relRangeArray[k].isWrite     = p->blklst[rbx].isWrite;

   rbx++;
   p->hint[nth].lstblkreleased++;
}
ghint.marh.relRangeCnt = k;
```

The next statements do some book keeping chores. The first one handles the case where GPFS_MAX_RANGE_COUNT divides the number of blocks touched. The second one handles an unusual situation that, after repeated testing, has never occurred, but that we cannot rule out. It is possible that we have not had hints accepted for some period of time and the records have been accessed and released (releasing un-issued hints is not a problem). Thus, state variables are set to be certain that we do not issue these hints since they are no longer needed.

```
if (p->hint[nth].lstblkreleased >= p->hint[nth].nblkstouched)
   relDone = TRUE;

if (p->nxtblktoissue <= rbx)
   p->nxtblktoissue = p->nbleh * (nth + 1);
```

This following while loop prepares the accRangeArray structures for issuing hints. It does this by extracting the block information from p->blklst starting with p->nxtblktoissue. Notice the variable rem; it is used to remember where we started so that if some of the blocks are not accepted, we are able to backtrack. As in the case of releasing blocks, if the number of blocks per record exceeds GPFS_MAX_RANGE_COUNT, it must cycle through the outer while loop again.

```
rem = p->nxtblktoissue;
ibx = p->nxtblktoissue;
k   = 0;
```

```
while (!accDone                 &&
       k < GPFS_MAX_RANGE_COUNT &&
       ibx < p->UBnblks           )
{
   if (p->blklst[ibx].blkoff >= 0)     /* Is the next entry OK? */
   {
      ghint.marh.accRangeArray[k].blockNumber = p->blklst[ibx].blknum;
      ghint.marh.accRangeArray[k].start       = p->blklst[ibx].blkoff;
      ghint.marh.accRangeArray[k].length      = p->blklst[ibx].blklen;
      ghint.marh.accRangeArray[k].isWrite     = p->blklst[ibx].isWrite;

      ibx++;
      k++;
   }
   else                            /* If not, find next good one. */
      ibx++;
}
ghint.marh.accRangeCnt = k;
p->nxtblktoissue       = ibx;
```

Here we set the fields of the gpfsFcntlHeader_t structure and call gpfs_fcntl() to release accessed blocks and issue the new hints that have been packaged in the preceding nested loops.

```
ghint.hdr.totalLength   = sizeof(ghint);
ghint.hdr.fcntlVersion  = GPFS_FCNTL_CURRENT_VERSION;
ghint.hdr.fcntlReserved = 0;
gpfs_fcntl(p->fd, &ghint);
```

Finally, there is one last set of book keeping chores to settle. If GPFS did not accept all of the hints that were issued, it is necessary to determine which ones were not accepted and set state variables to pick up where it left off. The variable nhntacc is set to the number of hints not accepted. rem records the first block issued in the most recent call to gpfs_fcntl(). This information is used to reset p->nxtblktoissue to the first unaccepted block so that we can start where we left off next time. Notice the test p->blklst[++ibx].blkoff < 0; because the record size is variable, the block offset blkoff is set to -1 for unused blocks in p->blklst.

```
nhntacc = ghint.marh.accRangeCnt;
if (nhntacc < GPFS_MAX_RANGE_COUNT)
{
   accDone = TRUE;           /* no more hints this time */

   ibx = rem - 1;
   k   = 0;
   while (k < nhntacc)
   {
      if (p->blklst[++ibx].blkoff < 0)    /* find next real data */
```

```
                continue;
             k++;
          }
          p->nxtblktoissue = ibx + 1;
       }
    }

    return 0;
}
```

Incidentally, when executing the GMGH code listed in Appendix G, “Benchmark and Example Code” on page 237, an option exists to print illustrative hint patterns which can clarify the actions of gmgh_issue_hint(). It is turned on by compiling the code with the -DDEBUG flag.

The process of issuing hints using the native MAR hint API is tedious, but it can be packaged in a more convenient API for use by application programmers as described next.

8.5.2 GMGH: A generic middle layer GPFS hints API

GMGH is a generic middle layer utility implementing a hints API for use by high level application programs. It is used in the benchmarks sited elsewhere in this book and it serves as an example for application programmers who can use it as is or adapt to their situation. To facilitate its role as an example, the GMGH code has been thoroughly documented. The code listed in Appendix G, “Benchmark and Example Code” on page 237 has been tested in both sequential and parallel programs; the code in this redbook is used only in sequential programs. This section describes many of the features of this API. By reading the preceding subsection and this subsection, and examining the source code in the appendix, the reader can quickly discern how it works internally.

The GMGH API calls

The GMGH API calls can be divided into two groups. The first group is intended to be used publicly by the application program, while the second group is intended for use privately within the GMGH utility only[4].

- Public (used by application program)
  - gmgh_init_hint()
  - gmgh_post_hint()
  - gmg_declare_1st_hint()
  - gmgh_xfer()

[4] This code is written in ANSI C; had it been written in C++, these functions would have been declared public and private respectively.

- Private (used within GMGH only)
  - gmgh_gen_blk()
  - gmgh_issue_hint()
  - gmgh_cancel_hint()

Each of these public functions are discussed individually. A general discussion that illustrates how to use them is mixed in with example code segments. The actions of the private functions are discussed in the context of their use by the public functions. Many references are made to the discussion associated with gmgh_issue_hint() example in the preceding subsection.

gmgh_init_hint()

This function initializes the structures used internally by GMGH. Its syntax is:

```
#include "gmgh.h"
int gmgh_init_hint(gmgh *p, int fd, int maxrsz, int maxhint)
```

The parameters are defined as follows:

- p

  p points to the GMGH structure defined in gmgh.h. The application program does not need to explicitly reference any of the fields in p.

- fd

  fd is the file descriptor for the opened file to which hints are being applied.

- maxrsz

  maxrsz specifies the maximum size in bytes that a record can be. Records can be variable in size in GMGH. So, in order to be processed, maxrsz must be at least as large as the largest record. If a record exceeds the size of maxrsz is accessed, blocks corresponding to the record beyond maxrsz will be ignored.

- maxhint

  maxhint specifies the maximum size of the hint set. A hint set is the set of records that is posted in the hint vector p->hint, whose corresponding blocks in p->blklst will later be issued as hints. maxhint must not exceed MAXHINT, as defined in gmgh.h; doing so will cause the function to fail, but the value of MAXHINT is somewhat arbitrary and can be changed if necessary (in our examples, its value is 128). It is customary to issue new hints after each call to read() or write(). Since the number of hints that will be accepted is not known, the hint set must be large enough so that there is always a set of blocks available for the issuing of new hints. Once the hint set is exhausted, it must be re-populated.

Setting the hint set too small prevents sufficient numbers of hints from being issued and increases overhead by forcing the hint set to be re-populated unnecessarily. Setting the hit set too large just wastes memory reserved for internal structures.

Upon success, this function returns 0, otherwise it returns -1.

gmgh_post_hint()

This function posts records indicating that they will be issued as hints in the future. The posting action places each posted record in the hint set (i.e., enters them into p->hint). As the record is posted, gmgh_gen_blk() is called by gmgh_post_hint() to determine how many blocks each record spans and it creates one entry for each block corresponding to the record in p->blklst. Before any of the records in the hint set are accessed, gmgh_post_hint() is called up to maxhint times posting records one after another. The syntax is:

```
#include "gmgh.h"
int gmgh_post_hint(gmgh *p, offset_t soff, int nb, int nth, int isWrite)
```

The parameters are defined as follows:

- p

  p points to the gmgh structure defined in gmgh.h.

- soff

  soff is the seek offset for the record to be accessed.

- nb

  nb is the actual length of the record to be accessed. nb < maxrsz.

- nth

  nth is an ordinal number of the record. It simply acts as an index into p->hint where the record is posted. When the record is actually accessed by gmgh_xfer(), nth is used again to correlate the record with blocks in p->blklst for releasing.

- isWrite

  isWrite is used to specify whether the operation is a read or write. isWrite = 1 (true) implies it is write, isWrite = 0 (false) implies it is a read.

Notice that there is no parameter for a buffer to contain the record; it is not needed at this point. The only information needed for posting a record and issuing hints for the corresponding blocks is the seek offset and size of the record.

The following is a simple example illustrating how to post hints. The loop generates the hint set one record at a time (next_rec() determines the seek offset for the next record). Notice that no records are accessed in the loop; that happens in a loop that comes after this one.

```
for (i = 0; i < maxhint; i++)
{
   soff = next_rec();
   gmgh_post_hint(p, soff, bsz, i, 1);
}
```

gmgh_declare_1st_hint()

gmgh_xfer() is used to access records and gmgh_issue_hint() is called after a record is accessed. But, the first time gmgh_xfer() is called for the hint set, no hints have been issued prior to accessing that first record. gmgh_declare_1st_hint() is therefore designed to be called after posting the last record in the hint set and prior to calling gmgh_xfer() to access the first record in the hint set. gmgh_declare_1st_hint() first calls gmgh_cancel_hint() to clean the slate (n.b., it releases any issued hints leftover from the previous hint set), then calls gpfs_issue_hint(). The syntax is:

```
#include “gmgh.h”
int gmgh_declare_1st_hint(gmgh *p)
```

- p

  p points to the gmgh structure defined in gmgh.h.

gmgh_xfer()

gmgh_xfer() is used to access records by calling the read() or write() system calls, but it first calls llseek() with the whence parameter set to SEEK_SET. After the record has been accessed, gmgh_issue_hint() is called which issues, re-issues and releases hints as necessary. gmgh_xfer() is called for each record in the hint set after all the records have been posted by gmgh_post_hint() and gmgh_declare_1st_hint() has been called the first time. The syntax is:

```
#include “gmgh.h”
int gmgh_xfer(gmgh *p, char *buf, int nth)
```

- p

  p points to the gmgh structure defined in gmgh.h.

- buf

  buf points to a buffer supplied by the user to contain the record. Its size must be at least maxrsz bytes.

- nth

nth is the ordinal number of the record and is an index into p->hint where the record is posted. gmgh_xfer() uses the nth entry in p->hint to get the seek offset, record size and whether it should be read or written.

General discussion and example

The following code segments are adapted from one of the benchmark programs referenced in Appendix G, “Benchmark and Example Code” on page 237 to illustrate how to use GMGH. This program can also be downloaded (see Appendix H, “Additional material” on page 259 for further details). In order to simplify the code segments and ensuing discussion, constants are used to initialize program parameters, error test and timing statements are removed, and self documenting identifiers are used.

To begin with, the GMGH header file containing function prototypes, other include files, constant definitions, and so forth, is included.

```
#include “gmgh.h”
```

The following statements are variable declarations. The variable names and associated comments should make their meaning clear.

```
int      fd;                          /* file descriptor */
int      nrec = NUMBER_OF_RECORDS;    /* obvious */
int      bsz  = SIZE_OF_RECORDS;      /* fixed size records */
char     buf[SIZE_OF_RECORDS];        /* record buffer */
int      nrn;                         /* next record number */
offset_t soff;                        /* seek offset */
gmgh     *p;                          /* pointer to gmgh structure */
int      maxhint = MAXHINT;           /* max number of hints */
int      i, k, nhint;                 /* miscellaneous */
```

The next two statements are some initialization tasks. Since we are generating a random I/O access pattern, we need a random number generator. We are calling it rand(); it returns single precision, floating point, uniform random deviates between 0 and 1. seed_rand() seeds the random sequence. Afterwords, we open a file to be written to whose size can exceed 2 GB.

```
seed_rand(1.0);                       /* seed the random number generator */

fd = open(“my_file”, O_RDWR | O_CREAT | O_TRUNC | O_LARGEFILE, 0777);
```

What comes next is the first GMGH call. We mallocate memory for the GMGH structure, initialize the fields to zero and call gmgh_init_hint().

```
p  = (gmgh*)malloc(sizeof(gmgh));
memset(p,’\0’,sizeof(gmgh));
gmgh_init_hint(p, fd, bsz, maxhint); /* Building hint data */
```

The following loop cycles through all of the records in chunks of size `maxhint` which sets the maximum size of the hint set. `nhint` is the actual size of the hint set; typically it is `maxhint`, but if `maxhint` does not divide `nrec`, then the last time through the loop, `nhint` is set to the appropriate size.

```
for (k = 0; k < nrec; k+=maxhint)
{
   if (k + maxhint > nrec) nhint = nrec - k;
   else                    nhint = maxhint;
```

The following code segment is the first of two inner loops. The next record number, nrn, is generated randomly. Based on the seek offset, soff is calculated. The purpose of this loop is to post hints for each record in the hint set. Immediately following the loop, we declare (i.e., issue) the first hint.

```
   for (i = 0; i < nhint; i++)
   {
      nrn  = (int)((float)nrec * rand());   /* 0.0 <= rand() < 1.0 */
      soff = (offset_t)nrn * (offset_t)bsz;

      gmgh_post_hint(p, soff, bsz, i, 1);   /* remember, 1 means write */
   }
   gmgh_declare_1st_hint(p);
```

The next inner loop actually writes the data using gmgh_xfer() after generating the data. make_data() generates the data to be written and uses the seek offset to be sure it generates the right data. Since the records are posted before the data is generated, it is important that correct data is placed in the buffer before writing it. But generating the data after posting it may not always be possible due to the difficulty of correlating the seek offset with the data (the seek offset may need to be recalculated). In that case, the application program can generate, save, and post the records in one loop and write them in the next loop using gmgh_xfer(). This requires extra memory (but no significant extra time) and allows the hints to be issued.

Also, remember that the hints are issued, re-issued, and released as part of the activities of gmgh_xfer().

```
   for (i = 0; i < nhint; i++)
   {
      make_data(soff, buf);
      gmgh_xfer(p, buf, i);     /* write the record */
   }
}
```

Benchmark results

Table 8-6 compares the benchmark codes of MAR hints in use versus MAR hints not in use. These are the I/O intensive versions of the benchmarks. Similar results hold for mixed CPU and I/O benchmarks. As can be readily seen from inspecting this table, the use of hints significantly improves the performance of the read benchmarks. What is puzzling is that performance is degraded for all of the write benchmarks except for the medium record size case (and here the improvement is marginal). The authors have tested parallel versions of these same benchmark programs on SP systems with VSD servers using small record random writes and the performance increased threefold.

Table 8-6 Using and not using MAR hints

Record size-Pattern	Write - NO hints	Write - using hints	Read - NO hints	Read - using hints
L-Random	64.40 MB/s	47.20 MB/s	29.80 MB/s	60.60 MB/s
M-Random	52.60 MB/s	56.00 MB/s	13.20 MB/s	55.20 MB/s
S-Random	6.23 MB/s	4.92 MB/s	1.39 MB/s	8.93 MB/s

In general, it is fair to conclude that random I/O access patterns are not as efficient as other access methods. Moreover, the relationships between the various solution strategies are somewhat perplexing. If a random pattern must be adopted in an application, it is recommended that careful benchmarking be done to assess its performance relative to these various strategies before implementing a definitive algorithm for use in production.

8.6 Multi-node performance

It is unlikely that a multi-node system will only have a single job executing on it at a time as is the case with the benchmarks discussed in Section 8.4, “Analysis of I/O access patterns” on page 154. A GPFS file system is designed to be accessible from multiple nodes having multiple jobs running simultaneously. But this suggests the question, how well do multiple programs simultaneously using the file system perform? To assess this question, the authors executed a series of benchmarks which simultaneously ran multiple programs (at most, one per node) and measured the I/O performance in MB/s. The results are summarized in Figure 8-3 on page 176.

In this series of tests, a sequential I/O access pattern using large records (e.g., 1024 KB with 256 KB blocks) is tested. The same benchmark programs and parameters are used as with the benchmarks in Section 8.4, “Analysis of I/O access patterns” (n.b., they are not parallel programs, just multiple instances of the same sequential program accessing a unique file each time).

Write tests are first launched on two, three and four nodes, followed by read tests on two, three and four nodes. Figures from single node tests are included in the table for comparison. In addition, tests mixing reads and writes are launched. In the first case, there is one write test launched on one node and one read test launched on another node. The test is repeated except that there are two write tests and two read tests launched with one test per node. Perusing this table, the reader sees that the I/O rate per node goes down, but more importantly that the aggregate I/O rate increases as the number of nodes increases (n.b., it drops off slightly on four nodes). Thus, more total work is being done.

		Number of nodes		
	1	2	3	4
Write-Test				
Aggregate Rate	66.97	105.36	120.10	110.90
Rate on node 1	66.97	52.74	41.12	28.77
Rate on node 2		52.68	40.03	27.73
Rate on node 3			40.14	28.69
Rate on node 4				28.88
Harmonic Mean	66.97	52.71	40.42	28.51
Read-Test				
Aggregate Rate	75.91	103.65	114.89	109.14
Rate on node 1	75.91	51.82	38.30	27.37
Rate on node 2		52.27	39.58	27.56
Rate on node 3			39.05	27.28
Rate on node 4				27.51
Harmonic Mean	75.91	52.04	38.97	27.43
Mixed Read/Write Test				
Aggregate Rate		92.11		117.11
Rate on node 1	write->	66.00	write->	41.04
Rate on node 2	read->	46.05	write->	41.77
Rate on node 3			read->	29.33
Rate on node 4			read->	29.28
Harmonic Mean		54.26		34.32

Figure 8-3 Multi-node tests

A similar tests were conducted using large (e.g., 1024 KB) and small (e.g., 16 KB) record random I/O access patterns and similar results were observed; the I/O rate per node goes down, but the aggregate I/O rate increases as the number of nodes increases.

8.7 Performance monitoring using system tools

In previous examples, the performance figures sited are based on application programming instrumentation. In particular, the programs contained calls to rtc(), which recorded times, and counters, which tracked the amount of data written. However, the user may wish to use system tools to monitor performance. Two such tools are iostat and filemon.

8.7.1 iostat

The **iostat** command is a useful command for monitoring I/O and CPU activity. It generates statistical reports at designated time intervals for designated devices and lists them on stdout. Since the focus of this book is on GPFS, we shall examine **iostat** regarding its disk I/O monitoring features.

The format of the **iostat** command for disk I/O is:

```
# iostat -d [physical volume] [interval [count] ]
```

The -d option specifies that a disk utilization report is generated. Each of the parameters in brackets are optional, but if missing have default values.

- `[physical volume]`

  This is a space separated list of physical volumes. If it is omitted, then all volumes are monitored.

- `[interval [count] ]`

  interval specifies a time interval in seconds between reports; if it is omitted, then only one report is generated, but if it is included, count can be given to specify how many reports are generated. If interval is specified, but count is not, the reports are generated indefinitely.

The statistics are reported in five columns for the -d option. The first report is the cumulative calculation of each of these statistics since the system was last rebooted. Each subsequent report is over the most recent time interval.

- tm_act - percentage of time the physical volume was active
- Kbps - I/O rate in KB per second
- tps - the number of transfers per second
- Kb_read - total KB read
- Kb-wrtn - total KB written

In the following example, each report consists of three rows, one for each listed physical volume. The first report is the cumulative measurement of each statistic since the system was last rebooted (n.b., the system used in this example had been recently rebooted). The second report was generated during a write benchmark and the third report during a read benchmark. Care should be taken interpreting these results. For example, I/O rate (Kbps) is for one physical volume only; it is not an aggregate I/O rate.

```
host1t:/> iostat -d hdisk3 hdisk4 hdisk5 60 3
Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk4           0.2      13.6       0.2     695227   1684270
hdisk3           0.2      13.5       0.2     691926   1673121
hdisk5           0.2      13.5       0.2     692550   1672352
hdisk4          46.3     4376.5      34.2          0     43776
hdisk3          44.9     4376.5      34.2          0     43776
hdisk5          45.9     4376.5      34.2          0     43776
hdisk4          47.9     4555.7      35.6      45568         0
hdisk3          43.6     4581.3      35.8      45824         0
hdisk5          47.1     4606.8      36.0      46080         0
```

8.7.2 filemon

filemon is another useful command for monitoring I/O activity. Unlike the **iostat** command which only examines file I/O from the perspective of physical volumes, **filemon** monitors the performance of the file system on behalf of logical files, virtual memory segments, logical volumes and physical volumes. In its normal mode, **filemon** runs in the background while application programs and system commands are being executed and monitored. It starts collecting data as soon as it is launched and stops once directed to by the **trcstop** command. There are many alternatives for using **filemon**, but we restrict this discussion to a limited number of them.

The format of **filemon** for the options we are considering is:

```
# filemon [-u] [-o File] [-O levels]
```

Each of the parameters in brackets are optional, but if missing, have default values.

- `[-u]`

  Collect statistics on files opened prior to launching the **filemon** command.

- `[-o File]`

  Redirect output to the named file. If omitted, output is sent to stdout.

- `[-O levels]`

  Collect statistics for the designated file system levels. The options are

– logical file (lf)
– logical volume (lv)
– virtual memory (vm)
– physical volume (pv)

If omitted, statistics are collected for the vm, lv and pv options.

After executing the **filemon** command, you must execute the **trcstop** command to stop the statistics being collected and to write the data to the output file. By default, the statistics are reported for the 20 most active files.

The following example illustrates how to use this tool. Statistics are being collected for the lf option. Two of the I/O benchmarks referenced in Appendix G, “Benchmark and Example Code” on page 237 are being executed and monitored. However, any application program or system command would do.

```
host1t:/> filemon -u -o fmon.out -O lf
Enter the "trcstop" command to complete filemon processing
host1t:/> ibm_sgw /gpfs1/fmeg 262144 8192 no seq
host1t:/> ibm_sgr /gpfs1/fmeg 262144 8192 no seq
host1t:/> trcstop
host1t:/> [filemon: Reporting started]
[filemon: Reporting completed]
[filemon: 137.177 secs in measured interval]
host1t:/>ls -l f*
-rw-r--r--   1 root     system      7500 Feb 28 20:17 fmon.out
```

The report in `fmon.out` is to lengthy to include in this example, so selected portions extracted and listed below. To begin with, only the five of the 20 `Most Active Files` are listed; the rest are omitted. The columns reporting statistics in this section represent (relative to the duration of the report):

- #MBs - MB per second
- #opns - number of times the file was opened
- #rds - number read system calls made against the file
- #wrs - number write system calls made against the file

After this, the first file in the Detailed File Stats section is listed.

These samples, from the fmon.out file, are enough to provide you with the flavor of the information provided in this helpful report. You should experiment with the other options.

```
host1t:/> less fmon.out
Wed Feb 28 20:15:20 2001
System: AIX host1t Node: 4 Machine: 000B4A7D4C00
Cpu utilization:  11.6%
Most Active Files
------------------------------------------------------------------------
```

```
  #MBs  #opns   #rds   #wrs  file                 volume:inode
---------------------------------------------------------------------------
4096.0      2   8192   8192  fmeg                 /dev/gpfs1:90129
  18.5      0    592      0  pid=19976_fd=5
   1.0      0    253     94  pid=24450_fd=4
   0.5      0    137      0  pid=14124_fd=0
   0.5      0    131     51  pid=14726_fd=4

---------------------------------------------------------------------------
Detailed File Stats
---------------------------------------------------------------------------
FILE: /gpfs1/fmeg  volume: /dev/gpfs1  inode: 90129
opens:                  2
total bytes xfrd:       4294967296
reads:                  8192    (0 errs)
  read sizes (bytes):   avg 262144.0 min  262144 max  262144 sdev     0.0
  read times (msec):    avg   3.353 min   2.930 max  34.183 sdev   0.554
writes:                 8192    (0 errs)
  write sizes (bytes):  avg 262144.0 min  262144 max  262144 sdev     0.0
  write times (msec):   avg   3.648 min   3.166 max 244.538 sdev   5.472
lseeks:                 16384
```

8.8 Miscellaneous application programming notes

This section discusses several smaller, but important items involving application programs using GPFS.

8.8.1 File space pre-allocation and accessing sparse files

A common task in some applications is to pre-allocate the space for a file and perhaps initialize all of the space to zeros. This can be quite expensive for large files. For example, a common task in some applications is to read a record from a file, add a new record to it, and write it back to the same location in the file.

There is a very quick and easy way to do this for a GPFS file. A program can write a record (or even just one byte) at the end of the file (find an example of this in the benchmark codes ibm_sgw and ibm_shw sited in Appendix G, “Benchmark and Example Code” on page 237). This pre-allocates all of the file space preceding this record and initializes it to zero. The operation takes something on the order of a millisecond or less to do. However, this is only pre-allocates virtual space.

For example, suppose a file system has been configured to have 1024 GB of storage and 960 GB is already occupied; there is only 64 GB of free space. Suppose job1 pre-allocates 50 GB of storage and actually writes 10 GB of data to the file. Doing an **ls -l** after job1 completes shows the file size to be 50 GB while a **du -k** shows its real size to be a little over 10 GB (remember, **du** also includes a file's indirect blocks; see Example 2-1, "Sparse file" on page 22).

Now suppose job2 begins, and it writes 50 GB of data to disk. The 50 GB of virtual space job1 allocated *plus* the 50 GB of space job2 actually use exceeds the 64 GB of free space that was available before job1 began. The reason that this can happen is that only 10 GB real space was used by job1. That means between job1 and job2, 60 GB of real space was used and now only 4 GB of file space is actually available in this file system. If job1 resumes activity and tries to use its remaining 40 GB of virtual space, the file system will run out of space.

This example illustrates an inconvenience associated with pre-allocating file space. In spite of this, many shops find that the time savings of file pre-allocation outweigh this inconvenience; they simply adjust their storage policies to handle this situation.

Continuing with this example (but assuming there is an abundance of file space), suppose a job reads a record of virtual file space that has not actually had anything written to it. Then the record is returned containing all zeros; this is a normal situation and not an error[5]. On the other hand, suppose a program tried to read a record beyond the end of the virtual space. This generates an error with the errno value EBADF (Bad file descriptor).

8.8.2 Notes on large files

GPFS and AIX, beginning in version 4.2, supports large files (i.e., file size exceeds 2 GB). This means that the seek offset value can be larger than a 32 bit signed integer can handle. To accommodate this change, new AIX I/O APIs were created. For example, such familiar calls as open() are lseek() are replaced with open64() and lseek64(). Variables declared as off_t are replaced with off64_t or the long type. This interface, however, is not POSIX compliant.

In order to use the standard POSIX I/O API with large file support, the application code must define the _LARGE_FILES[6] flag before the inclusion of any header files. This automatically replaces POSIX I/O system calls and types with their 64 bit equivalent. The easiest way to do this is to include this flag on the compile line

[5] But caution is urged when reading a record that contains both virtual and real space. The authors have occasionally encountered unexpected results in this situation.

[6] The _LARGE_FILES flag should not be confused with the open() system call flag O_LARGEFILE; this latter flag is used to open a large file when a file system like NFS or JFS may not be configured to support files exceeding 2 GB by default. The O_LARGEFILE flag doesn't effect GPFS since it is always configured to support files greater than 2 GB.

in the makefile (i.e., -D_LARGE_FILES). The benchmark and example codes referenced in Appendix G, “Benchmark and Example Code” on page 237 defines this flag in this way. See General Programming Concepts: Writing and Debugging Programs , 2nd edition, Sept 1999 for further details.

8.8.3 GPFS library

When using the MAR hint API, it is necessary to link the application program with libgpfs.a. This is done by placing the flag -lgpfs on the compile line in the makefile. This flag is not needed for programs accessing GPFS files.

9

Problem determination

This chapter is designed to assist with the problem determination in a GPFS for HACMP Cluster environment. It is intended to compliment the product specific guides for this environment, namely:

- *GPFS for AIX: Problem Determination Guide, GA22-7434*
- *HACMP/ES 4.4 for AIX: Installation and Administration Guide, Vol.2, SC23-4306*
- *HACMP/ES 4.4 for AIX: Troubleshooting Guide, SC23-4280*

This chapter will review:

- Logs available for HACMP and GPFS
- Group services (grpsvcs)
- Topology services (topsvcs)
- Disk related errors
  - varyonvg related problems
  - Definition problems
  - SSA Disk Fencing
    - Determining fence IDs
    - Setting fence IDs
    - GPFS daemon error messages when the fence ID is set incorrectly

- Internode communications
  - Checking interfaces defined to HACMP
  - .rhost files
  - Testing rsh/rcp for correct setup

9.1 Log files

This section will review the logs available for assisting in the problem determination process from both the HACMP and GPFS products.

In addition to the logs that are specific to both HACMP and GPFS, many errors result in the creation of an AIX error log entry. These error logs are reviewed with the **errpt** command. There are often helpful details in the AIX error logs that correspond to the information in the log files of HACMP or GPFS.

9.1.1 Location of HACMP log files

/usr/es/adm/cluster.log	Contains time-stamped, formatted messages generated by HACMP/ES scripts and daemons. This is a standard text file that can be viewed directly or with the cldiag utility.
/tmp/hacmp.out	Contains time-stamped, formatted messages generated by HACMP/ES scripts on the current day.
system error log	Contains time-stamped, formatted messages from all AIX subsystems, including scripts and daemons.

There are other logs specific to the other HACMP/ES subsystems including group services and topology services. These are detailed in the aforementioned HACMP manuals.

9.1.2 Location of GPFS log files

The following are two important log files:

/var/adm/ras/mmfs.log.latest → link to latest mmfs log file

/var/adm/ras/mmfs.log.previous → link to previous mmfs log file

In normal operations, GPFS writes both operational messages and error data to the mmfs log file. The log files are found in the /var/adm/ras directory on each node. The mmfs log file is called mmfs.log.date where date is the time-stamp of when the latest instance of GPFS was started on the node. The latest mmfs log can be found by using the symbolic link /var/adm/ras/mmfs.log.latest. The mmfs log from the previous instance of GPFS can be found by using the symbolic link /var/adm/ras/mmfs.log.previous.

By default, the last 11 days of logs are retained. This value can be adjusted by changing the value in the /usr/lpp/mmfs/bin/runmmfs script.

An example of the messages that appear in the mmfs log are:

```
Fri Apr  6 16:02:00 EDT 2001 runmmfs starting
Removing old /var/adm/ras/mmfs.log.* files:
Loading modules from /usr/lpp/mmfs/bin
GPFS: 6027-500 /usr/lpp/mmfs/bin/mmfs loaded and configured.
Fri Apr  6 16:02:01 2001: GPFS: 6027-310 mmfsd initializing.
Fri Apr  6 16:02:01 2001: GPFS: 6027-1531 useSPSecurity no
Fri Apr  6 16:02:02 2001: GPFS: 6027-841 Cluster type: 'HACMP'
Fri Apr  6 16:02:03 2001: Using TCP communication protocol
Fri Apr  6 16:02:03 2001: GPFS: 6027-1710 Connecting to 9.114.121.2
Fri Apr  6 16:02:03 2001: GPFS: 6027-1711 Connected to 9.114.121.2
Fri Apr  6 16:02:03 EDT 2001 /var/mmfs/etc/gpfsready invoked
Fri Apr  6 16:02:03 2001: GPFS: 6027-300 mmfsd ready
```

The GPFS daemon is not ready to accept commands until the message `GPFS: 6027-300 mmfsd ready` has been issued.

Note: The proper operation of both GPFS and HACMP depend on having space available for their logs files. The /usr, /var, and /tmp filesystems should be monitored to be sure that they do not become full.

9.2 Group Services

The successful startup of GPFS depends on group services being active on all nodes that are going to be part of the GPFS cluster. GPFS will not start on a node without group services being active on that node. Group Services is started on a node when HACMP/ES is started.

9.2.1 Checking the Group Services subsystem

Issue the **`lssrc -ls grpsvcs`** command to list the status of the group services subsystem. If the grpsvcs subsystem is not active on the node, then the command returns the following message:

```
(17:13:21) c185n01:/ # lssrc -ls grpsvcs
0513-036 The request could not be passed to the grpsvcs subsystem.
Start the subsystem and try your command again.
```

Also note that the hagsd daemon is not active on this node:

```
(17:16:31) c185n01:/ # ps -ef|grep grpsvcs|grep -v grep
(17:19:48) c185n01:/ #
```

The lack of group services means that HACMP is not active on this node. Start HACMP and run the same series of commands. How to start HACMP is covered in Chapter 5, “Configuring HACMP/ES” on page 71.

With HACMP active on a node, the hagsd daemon is also active:

```
(17:29:49) c185n01:/ #  ps -ef|grep grpsvcs|grep -v grep
    root 12532  7820   0 17:22:46      -  0:00 hagsd grpsvcs
(17:31:07) c185n01:/ #
```

Now that group services is active, the "lssrc -ls grpsvcs" command shows the activity of the groups that are subscribed to group services, in this case, HACMP.

```
(17:29:42) c185n01:/ # lssrc -ls grpsvcs
Subsystem         Group            PID     Status
 grpsvcs          grpsvcs          12532   active
3 locally-connected clients.  Their PIDs:
13946(hagsglsmd) 18770(haemd) 15244(clstrmgr)
HA Group Services domain information:
Domain established by node 5
Number of groups known locally: 3
                   Number of   Number of local
Group name         providers   providers/subscribers
ha_em_peers              8           1           0
CLRESMGRD_15             8           1           0
CLSTRMGR_15              8           1           0
(17:29:49) c185n01:/ #
```

The above display from the **lssrc -ls grpsvcs** command shows that this node is a "provider" of services, one of eight in this cluster.

The above display shows a state which is ready to support the operation of a GPFS cluster.

Operation of GPFS without Group Services

If the GFPS daemon is started before group services is active, then the daemon will not start. Rather then starting, GPFS will monitor the existence of group services, waiting for it to come active. An excerpt from the /var/adm/ras/mmfs.log.latest shows this behavior:

```
(18:11:16) c185n01:/ # tail -f /var/adm/ras/*latest
Sun Apr  8 18:11:06 EDT 2001 runmmfs starting
Removing old /var/adm/ras/mmfs.log.* files:
Loading modules from /usr/lpp/mmfs/bin
GPFS: 6027-506 /usr/lpp/mmfs/bin/mmfskxload: /usr/lpp/mmfs/bin/mmfs is already
 loaded at 90670232.
```

```
Sun Apr  8 18:11:07 EDT 2001 runmmfs: 6027-1242 GPFS is waiting for grpsvcs
Sun Apr  8 18:12:07 EDT 2001 runmmfs: 6027-1242 GPFS is waiting for grpsvcs
Sun Apr  8 18:13:08 EDT 2001 runmmfs: 6027-1242 GPFS is waiting for grpsvcs
Sun Apr  8 18:14:08 EDT 2001 runmmfs: 6027-1242 GPFS is waiting for grpsvcs
Sun Apr  8 18:15:08 EDT 2001 runmmfs: 6027-1242 GPFS is waiting for grpsvcs
```

After the group services daemon is started, then GPFS will continue with it's initialization. An excerpt from the /var/adm/ras/mmfs.log.latest shows this behavior:

```
(18:11:16) c185n01:/ # tail -f /var/adm/ras/*latest
Sun Apr  8 18:11:06 EDT 2001 runmmfs starting
Removing old /var/adm/ras/mmfs.log.* files:
Loading modules from /usr/lpp/mmfs/bin
GPFS: 6027-506 /usr/lpp/mmfs/bin/mmfskxload: /usr/lpp/mmfs/bin/mmfs is already
 loaded at 90670232.
Sun Apr  8 18:11:07 EDT 2001 runmmfs: 6027-1242 GPFS is waiting for grpsvcs
Sun Apr  8 18:12:07 EDT 2001 runmmfs: 6027-1242 GPFS is waiting for grpsvcs
Sun Apr  8 18:13:08 EDT 2001 runmmfs: 6027-1242 GPFS is waiting for grpsvcs
Sun Apr  8 18:14:08 EDT 2001 runmmfs: 6027-1242 GPFS is waiting for grpsvcs
Sun Apr  8 18:15:08 EDT 2001 runmmfs: 6027-1242 GPFS is waiting for grpsvcs
Sun Apr  8 18:16:03 2001: GPFS: 6027-310 mmfsd initializing.
Sun Apr  8 18:16:03 2001: GPFS: 6027-1531 useSPSecurity no
Sun Apr  8 18:16:04 2001: GPFS: 6027-841 Cluster type: 'HACMP'
Sun Apr  8 18:16:07 2001: Using TCP communication protocol
Sun Apr  8 18:16:07 2001: GPFS: 6027-1710 Connecting to 9.114.121.2
Sun Apr  8 18:16:07 2001: GPFS: 6027-1711 Connected to 9.114.121.2
Sun Apr  8 18:16:07 EDT 2001 /var/mmfs/etc/gpfsready invoked
Sun Apr  8 18:16:07 2001: GPFS: 6027-300 mmfsd ready
```

9.3 Topology Services

The operation of GPFS depends on the availability of the network that has been defined for its use. GPFS subscribes to topology services in order to monitor the health of this network. Like group services, the topology services daemon (hatsd) is started by HACMP.

9.3.1 Checking the Topology Services subsystem

Issue the **`lssrc -ls topsvcs`** command to check on the operational status of the topology services subsystem. Here is an example of the topology services status with the network named GBether is the GPFS network.

```
(18:00:25) c185n01:/ # lssrc -ls topsvcs
Subsystem         Group            PID     Status
 topsvcs          topsvcs          10458   active
Network Name   Indx Defd Mbrs St Adapter ID      Group ID
GBether_0      [ 0]    8    8  S 9.114.121.1     9.114.121.8
GBether_0      [ 0] en1          0x32d0d6a5      0x32d0d6be
HB Interval = 1 secs. Sensitivity = 4 missed beats
  2 locally connected Clients with PIDs:
haemd( 18770) hagsd( 12532)
  Dead Man Switch Enabled:
     reset interval = 1 seconds
     trip  interval = 8 seconds
  Configuration Instance = 3
  Default: HB Interval = 1 secs. Sensitivity = 4 missed beats
  Daemon employs no security
  Data segment size: 6764 KB. Number of outstanding malloc: 318
  User time 1 sec. System time 1 sec.
  Number of page faults: 2. Process swapped out 0 times.
  Number of nodes up: 8. Number of nodes down: 0.
(18:04:40) c185n01:/ #
```

The above status show the network GBether that has eight members and this nodes state (the "St" field) is S.

Topology Services Display with Network Interface Problems

If the interface associated with the GPFS network was to go down then the topology services status would change reflecting this operational state.

For example, on node c185n01, the interface in the network GBether is en1. If it was brought down by the **`ifconfig en1 down`** command, a number of things would happen on this node. First, the operational status of the network would change in topology services. Secondly, as GPFS is dependent on this network for it's internode communication, the GPFS daemon would go down.

Here is the output from **`lssrc -ls topsvcs`** after en1 was brought down:

```
(18:21:28) c185n01:/ # lssrc -ls topsvcs
Subsystem         Group            PID     Status
 topsvcs          topsvcs          15306   active
Network Name   Indx Defd Mbrs St Adapter ID      Group ID
GBether_0      [ 0]    8    0  D 9.114.121.1
GBether_0      [ 0] en1
HB Interval = 1 secs. Sensitivity = 4 missed beats
  2 locally connected Clients with PIDs:
haemd( 15478) hagsd( 18798)
  Dead Man Switch Enabled:
```

```
    reset interval = 1 seconds
    trip  interval = 8 seconds
  Configuration Instance = 3
  Default: HB Interval = 1 secs. Sensitivity = 4 missed beats
  Daemon employs no security
  Data segment size: 6764 KB. Number of outstanding malloc: 316
  User time 0 sec. System time 0 sec.
  Number of page faults: 1. Process swapped out 0 times.
  Number of nodes up: 8. Number of nodes down: 0.
```

Note the change in the "St" field from "S" to "D".

Another way to view this interface is with the **netstat -in** command:

```
(18:21:33) c185n01:/ # netstat -in
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
en0   1500  link#2      0.60.94.e9.7f.84    625423     0   616640     0     0
en0   1500  9.114.121.6 9.114.121.65        625423     0   616640     0     0
en1*  1500  link#3      0.4.ac.7c.d6.c0     497283     0   589281     0     0
en1*  1500  9.114.121   9.114.121.1         497283     0   589281     0     0
lo0   16896 link#1                         439176     0   442219     0     0
lo0   16896 127         127.0.0.1           439176     0   442219     0     0
lo0   16896 ::1                           439176     0   442219     0     0
```

Note the asterisk (*) next to the interface en1 marking it down.

In addition to topology services changing states in response to en1, GPFS has also been informed by its membership that en1 has gone down. Here is the mmfs log showing this state change:

```
Sun Apr  8 18:56:00 2001: GPFS: 6027-300 mmfsd ready
Sun Apr  8 18:56:31 2001: GPFS: 6027-820 My adapter en1 has failed. Shutting
 down the daemon.
Sun Apr  8 18:56:31 2001: GPFS: 6027-831 Terminated connections to Group
 Services.
Sun Apr  8 18:56:31 2001: GPFS: 6027-650 The mmfs daemon is shutting down
 abnormally.
Sun Apr  8 18:56:31 2001: GPFS: 6027-311 mmfsd shutting down.
```

9.4 Disk problem determination

This version of GPFS is limited to using SSA disks when running on HACMP clusters. The following section will deal with typical problems that may be encountered in the operation of these SSA multi-tailed clusters.

The process of defining volume groups, defining logical volumes, getting the volume groups imported to all the nodes in the GPFS cluster, setting the volume group "AUTO ON" parameter is a complicated, but critical procedure for the proper operation of a GPFS cluster environment.

Chapter 6, “Configuring GPFS and SSA disks” on page 91 discusses how to define volume groups and logical volumes and how to use **importvg** to import these volume groups to the rest of the nodes in the GPFS cluster. The problems that may be encountered if these procedures are not followed exactly will be discussed In the coming sections, as well as procedures that can be used to verify that all the disks, volume groups, and logical volumes are in the proper state for GPFS usage.

9.4.1 GPFS and the varyonvg command

varyonvg is the AIX command to make a volume group available to a node. In the GPFS for HACMP clusters environment the same physical volume must be available to all the nodes in the cluster concurrently. The technique used is to **varyonvg** each volume group with the -u option. The **varyonvg -u** option allows a volume group to be varied on to a node without a lock being set on that volume group. Contrast this behavior to a normal (no -u parameter) **varyonvg** command, where the node that varies on the volume group locks it with the expressed purpose of keeping other nodes from accessing it concurrently.

If a volume group is varied online to a node without using the -u option of the **varyonvg** command, then all other nodes will lose access to the volume group. This includes nodes that had access to the volume through the proper use of the **varyonvg** command with the -u option.

We recommend **varyonvg** commands not be issued by users and to allow GPFS to manage the varyonvg state of all of it's volume groups. GPFS is capable of varying on disks for any operation, creation of file systems (**mmcrfs**), replacement of disks in file systems (**mmrpldisk**), adding disks to existing file systems (**mmadddisk**), or the mounting of file systems via the **mount** command. There is never a case where a user needs to issue a **varyonvg** command for a volume group for GPFS's usage.

Problems encountered during varyonvg processing

One potential error during the definition of a disk for usage in the GPFS for HACMP clusters environment is not to set the volume group's AUTO ON parameter to NO. This parameter is properly set to NO by the -n option of the **mkvg** command. The AUTO ON parameter is also changed on each node after the volume group has been imported with the **chvg -a n** command.

If this parameter is not properly set, then the default behavior of the AUTO ON parameter is to vary on the volume group at boot time. In the case where one or more nodes have this parameter improperly set, then one node will get the volume group varied online and all the other nodes will have errors trying to read the volume group.

Typical errors that will be seen will include:

```
(21:34:00) c185n01:/ # varyonvg -u gpfsvg16
PV Status:      hdisk66 000167498707c169        PVNOTFND
0516-013 varyonvg: The volume group cannot be varied on because
        there are no good copies of the descriptor area.
```

This is typical of the node where this **varyonvg** command is running from being locked out from a disk that is in an improper vary on state on another node.

There is an SSA utility command called **ssa_rescheck** that can assist with this problem state.

```
(21:34:30) c185n01:/ # ssa_rescheck -l hdisk66
  Disk     Primary  Secondary  Adapter  Primary Secondary Reserved
           Adapter  Adapter    In Use   Access  Access    to
  hdisk66  ssa0     ====       ssa0     Busy    ====      c185n02
```

The **ssa_resehck** command is in the devices.common.IBM.ssa.diag fileset. Its full usage is documented in the manual: Advanced SerialRaid Users Guide and Maintenance Information, SA33-3285

In order to solve the problem of not having the proper access to a volume group, the action is to either run the **ssa_rescheck** command and determine which node had it in the wrong state, or to vary off the volume group from all nodes and re-try the operation that created the above failure.

9.4.2 Determining the AUTO ON state across the cluster

The AUTO ON setting for a volume group can be determined with the **lsvg** command:

```
(21:34:07) c185n02:/ # lsvg gpfsvg16
VOLUME GROUP:   gpfsvg16                VG IDENTIFIER:  000202569Ocaf44f
VG STATE:       active                  PP SIZE:        16 megabyte(s)
VG PERMISSION:  read/write              TOTAL PPs:      268 (4288 megabytes)
MAX LVs:        256                     FREE PPs:       0 (0 megabytes)
LVs:            1                       USED PPs:       268 (4288 megabytes)
OPEN LVs:       0                       QUORUM:         2
```

```
TOTAL PVs:      1                       VG DESCRIPTORS: 2
STALE PVs:      0                       STALE PPs:      0
ACTIVE PVs:     1                       AUTO ON:        no
Concurrent:     Capable                 Auto-Concurrent: Disabled
VG Mode:        Non-Concurrent
MAX PPs per PV: 1016                    MAX PVs:        32
```

Here the setting AUTO ON is set to no which is the proper setting for volume groups in the GPFS for HACMP clusters environment.

The **lsvg** command needs to have the volume group online before it returns information including the setting of the AUTO ON setting for that volume group.

Another command that can be used to determine the AUTO ON setting for a volume group without that volume group having to be online is the **getlvodm** command.

getlvodm -u volume_group_name returns a y or an n depending on the setting of AUTO ON.

Here is a sample script that will return all the settings of all volume groups on one node whose names include the string gpfs.

```
#!/usr/bin/ksh
#
for vg in $(lsvg |grep gpfs)
do
    auton=$(getlvodm -u $vg)
    echo VG:  $vg  AutoOn: $auton
done
```

Here is an example of the above script on a node within a cluster, assuming the above script is called chkautovg:

```
(00:33:29) c185n01:/ #  chkautovg
        VG: gpfsvg0 AutoOn: n
        VG: gpfsvg1 AutoOn: n
        VG: gpfsvg2 AutoOn: n
        VG: gpfsvg3 AutoOn: n
        VG: gpfsvg5 AutoOn: n
        VG: gpfsvg6 AutoOn: n
        VG: gpfsvg7 AutoOn: n
        VG: gpfsvg8 AutoOn: n
        VG: gpfsvg9 AutoOn: n
        VG: gpfsvg10 AutoOn: n
        VG: gpfsvg11 AutoOn: n
        VG: gpfsvg12 AutoOn: n
```

```
        VG: gpfsvg13 AutoOn: n
        VG: gpfsvg14 AutoOn: n
        VG: gpfsvg15 AutoOn: n
        VG: gpfsvg4 AutoOn: n
        VG: gpfsvg16 AutoOn: n
        VG: gpfsvg17 AutoOn: n
        VG: gpfsvg18 AutoOn: n
        VG: gpfsvg19 AutoOn: n
        VG: gpfsvg20 AutoOn: n
        VG: gpfsvg21 AutoOn: n
        VG: gpfsvg22 AutoOn: n
        VG: gpfsvg23 AutoOn: n
        VG: gpfsvg24 AutoOn: n
        VG: gpfsvg25 AutoOn: n
        VG: gpfsvg26 AutoOn: n
        VG: gpfsvg27 AutoOn: n
        VG: gpfsvg28 AutoOn: n
        VG: gpfsvg29 AutoOn: n
        VG: gpfsvg30 AutoOn: n
        VG: gpfsvg31 AutoOn: n
(00:33:36) c185n01:/ #
```

This would have to be run on each node within the cluster and the grep parameters may have to be adjusted according to the naming standards of the GPFS vgs in any individual cluster.

9.4.3 GPFS and the Bad Block relocation Policy

Another of the critical options that is documented on the creation of a logical volume is the -b option of the **mklv** command, the Bad Block relocation policy option.

In an AIX system, the Bad Block relocation policy defaults to “y,” which causes Bad Block relocation to occur. A high level description of what happens upon the discovery of a Bad Block (BB) in a logical volume by the logical volume manager (LVM) is that an alternate block is assigned in the place of the bad one, the data in the BB is copied to the new one (if possible) and this change to the logical volume is then recorded in the object data manager (ODM).

In the concurrently shared disk environment, Bad Block relocation cannot be allowed to happen as there is no coordination between nodes to communicate this Bad Block relocation activity. If a node relocated a block while other nodes were concurrently accessing the modified disk, the other nodes would continue to attempt to access the relocated block at its original position. At this point, data integrity would be compromised.

In order to prevent this situation from occurring, all logical volumes in the GPFS for HACMP clusters environment need to have Bad Block relocation turned off (The BB POLICY must be set to non-relocatable.).

To check the setting of Bad Block relocation use the **lslv** command:

```
(21:52:08) c185n02:/ # lslv gpfslv16
LOGICAL VOLUME:     gpfslv16               VOLUME GROUP:   gpfsvg16
LV IDENTIFIER:      0002025690caf44f.1     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       closed/syncd
TYPE:               mmfsha                 WRITE VERIFY:   off
MAX LPs:            268                    PP SIZE:        16 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                268                    PPs:            268
STALE PPs:          0                      BB POLICY:      non-relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: off
EACH LP COPY ON A SEPARATE PV ?: yes
```

9.4.4 GPFS and SSA fencing

SSA disk fencing is a facility that is provided in the SSA disk subsystem. It allows multiple using systems to control access to a common set of disks.

By using the SSA fencing commands, GPFS can prevent particular nodes from accessing particular disks while all other nodes can continue to have normal access. GPFS uses disk fencing primary in recovery situations, but its use is not limited to this area.

To use disk fencing, the node_number attribute of the ssar router must be set to a unique value for each node in the cluster. In the GPFS for HACMP clusters environment, the value that is set in the node_number attribute is the HACMP node number as returned by the **/usr/es/sbin/cluster/utilities/clhandle** command.

An example of the **clhandle** command:

```
(22:19:23) c185n01:/ # clhandle -a
1 c185n01
2 c185n02
3 c185n03
4 c185n04
5 c185n05
6 c185n06
```

```
7 c185n07
8 c185n08
```

This shows an eight node cluster; the nodes are numbered from one to eight. The value of the ssa router object ssar is determined by the **lsattr** command:

```
(22:20:52) c185n01:/ # gdsh lsattr -El ssar
c185en01: node_number 1 SSA Network node number True
c185en02: node_number 2 SSA Network node number True
c185en03: node_number 3 SSA Network node number True
c185en04: node_number 4 SSA Network node number True
c185en05: node_number 5 SSA Network node number True
c185en06: node_number 6 SSA Network node number True
c185en07: node_number 7 SSA Network node number True
c185en08: node_number 8 SSA Network node number True
```

This shows how the ssar node_number attribute has been set to be equal to the node number on each node in the cluster. The procedure to change the ssar node_number is the **chdev** command:

```
chdev -l ssar -a node_number=X
```

In the above example, X is the node number as returned by **clhandle**.

If there is an error returned:

```
(22:23:11) c185n01:/ # chdev -l ssar -a node_number=1
Method error (/usr/lib/methods/chgssar):
        0514-031 A device is already configured at the specified location.
```

The device ssar already configured the child devices and cannot be changed with the **chdev** command. It must be changed with the HACMP command, **set_fenceid**:

```
/usr/es/sbin/cluster/utilities/set_fence_id -l gpfslv16 1
```

In the above example, -l is a parameter that accepts a logical volume name. Provide it with a logical volume name that is on the SSA disks. The last parameter is the ssar node_number that is set to the fence ID.

SSAR Node Number and GPFS Startup

Since GPFS depends on SSA fencing to ensure file system data integrity, the GPFS daemon checks the value of the ssar node_number object on each node each time it is started. If the GPFS daemon finds that the node_number set in SSAR does not match the node number assigned to the node as returned by the **clhandle** command, the daemon will not start.

Here is an excerpt from the mmfs log of this event:

```
The fence id 9 returned from my_fence_id command does not match with my node
 number 1. First set fence id and try again
The fence id 9 returned from my_fence_id command does not match with my node
 number 1. First set fence id and try again
The fence id 9 returned from my_fence_id command does not match with my node
 number 1. First set fence id and try again
```

In this case, the ssar node_number is incorrectly set to nine while it should be set to one, to match this node's HACMP node number:

```
(22:38:50) c185n01:/ # clhandle
1 c185n01
```

Disks passed to GPFS with fence bits set

GPFS uses SSA fencing as a hardware technique to control which nodes can and cannot access disks. While GPFS manages the state of the fencing without any special interaction with system administrators, there is one case where GPFS will not unfence a disk in order to use it even though the disk has been passed to GPFS for use by an administrative command.

The case is where GPFS has been asked to use a disk and the disk is found to be fenced. The disk would have been a parameter to a GPFS administrative command like **mmcrfs**, **mmrpldisk** or **mmadddisk**. In this case, GPFS will issue an error message and not use this disk.

The reasoning behind this decision is that GPFS does not know why this disk is fenced and it may be fenced by another application. The GPFS response is to ask the administrator to ensure that this disk is, in fact, not in use by any other applications and to unfence it. Once the disk has been unfenced, it will then be eligible for use by GPFS.

The disk can be unfenced by the HACMP command:

```
/usr/es/sbin/cluster/utilities/fence_clear_all
```

9.5 Internode communications

GPFS communicates across the nodes in a cluster for a variety of reasons, including sharing file system state data between GPFS daemons and administrative commands keeping configuration files in sync. In order for this communication to happen properly in the GPFS for HACMP clusters environment, **rsh** and **rcp** must be setup to operate properly.

During the installation and configuration of HACMP/ES in standard security mode, the .rhosts files will normally be created and distributed to all nodes in the cluster. All of the interfaces that will be used by GPFS must be included in the file along with the root userID.

The .rhosts file must be in root's home directory, normally /. While the operation of **rsh** does not depend on any specific permissions on the .rhost file, it is recommended that it be set to 600, which is owner read/write only (600, rw-------).

If the rhosts file is missing, then the **rsh** command will fail. If the host issuing the command does not have its name in the target node's .rhosts file, the command will fail.

Example of failed commands:

```
(14:17:57) c185n01:/ # rsh c185n02 date
rshd: 0826-813 Permission is denied.
```

The same message is issued to the user if the target node is missing its .rhosts file, or if the .rhosts file exists but is missing the entry of the issuing node.

9.5.1 Testing the internode communications

In order to test the ability of all nodes to communicate with each other, issue the command:

```
(01:01:30) c185n01:/ # for i in $(cat /etc/cluster.nodes)
> do
> rsh $i date
> done
Fri Apr 20 01:01:48 EDT 2001
Fri Apr 20 01:02:06 EDT 2001
Fri Apr 20 01:02:41 EDT 2001
Fri Apr 20 01:01:34 EDT 2001
Fri Apr 20 01:01:31 EDT 2001
Fri Apr 20 01:02:16 EDT 2001
(01:01:48) c185n01:/ #
```

This will run the **date** command on each node in the nodeset. This command should be repeated on each node within the cluster. Any failures will indicate that there is a problem with the .rhost file on the node returning the error.

A

Mapping virtual disks to physical SSA disks

Once SSA disks are cabled and the hosts powered on, one-to-one relationships (mapping or translations) will be established between virtual (hdisk) and physical (pdisk) disk drives on all hosts that are connected via SSA adapter cards. Normally, we can get along with just knowing the hdisk number, but in the case of concurrently mounted drives, different host machines can refer to the same physical disk with a different hdisk address. Because of this, we need to become knowledgeable in determining which hdisk address refers to a predetermined physical disk known to all the hosts.

There are several ways to determine physical disks, but the most accessible is the PVID (Physical Volume ID). The PVID is the middle column listed by the **lspv** command. The PVID is unique for every SSA disk manufactured. Comparing the PVID's across nodes will show that hdiskx names are not always consistent between nodes.

SSA commands

To translate (or map) pdisk to hdisk we used **ssaxlate -l pdisk0** and hdisk to pdisk was **ssaxlate -l hdisk3**. This command has limited applications in our case but is useful for some verification.

```
host1t:/> ssaxlate -l pdisk0
hdisk3
host1t:/> ssaxlate -l hdisk7
pdisk4
```

Using diag for mapping

Contained under the **diag** facility in AIX resides a tool for viewing several different features of the SSA disks and cabling configuration, including the location of disks in the SSA drawers and their current status. When performing problem determination, this tool can make selected disk drives blink to aid in hardware identification. Enter **diag** on the command line and follow the appropriate path to desired information.

```
diag
   Task Selection(Diagnostics, Advanced Diagnostics, Service Aids, etc.)
      SSA Service Aids
```

```
Set Service Mode
  Link Verification
  Configuration Verification
  Format Disk
  Certify Disk
  Display/Download Disk Drive Microcode
  Link Speed
  Physical Link Configuration
  Enclosure Configuration
  Enclosure Environment
  Enclosure Settings
  SMIT - SSA RAID Arrays
  SMIT - SSA Disks
```

```
diag
   Task Selection(Diagnostics, Advanced Diagnostics, Service Aids, etc.)
      SSA Service Aids
         Configuration Verification
```

```
  host1t:pdisk0                    29CD5754  SSA160 Physical Disk Drive
  host1t:pdisk1                    29CD697E  SSA160 Physical Disk Drive
  host1t:pdisk2                    29CD6980  SSA160 Physical Disk Drive
  host1t:pdisk3                    29CD6A3B  SSA160 Physical Disk Drive
  host1t:pdisk4                    29CD6ACA  SSA160 Physical Disk Drive
  host1t:pdisk5                    29CD6B0D  SSA160 Physical Disk Drive
  host1t:pdisk6                    29CD6C12  SSA160 Physical Disk Drive
  host1t:pdisk7                    29CD6D07  SSA160 Physical Disk Drive
  host1t:pdisk8                    29CD6D0A  SSA160 Physical Disk Drive
  host1t:pdisk9                    29CD6E19  SSA160 Physical Disk Drive
  host1t:pdisk10                   29CD6EDB  SSA160 Physical Disk Drive
  host1t:pdisk11                   29CD6EE1  SSA160 Physical Disk Drive
  host1t:pdisk12                   29CD70BD  SSA160 Physical Disk Drive
  host1t:pdisk13                   29CD70C5  SSA160 Physical Disk Drive
  host1t:pdisk14                   29CD731D  SSA160 Physical Disk Drive
  host1t:pdisk15                   29CD7372  SSA160 Physical Disk Drive
  host1t:hdisk3                    29CD5754  SSA Logical Disk Drive
  host1t:hdisk4                    29CD697E  SSA Logical Disk Drive
  host1t:hdisk5                    29CD6980  SSA Logical Disk Drive
  host1t:hdisk6                    29CD6A3B  SSA Logical Disk Drive
  host1t:hdisk7                    29CD6ACA  SSA Logical Disk Drive
  host1t:hdisk8                    29CD6B0D  SSA Logical Disk Drive
  host1t:hdisk9                    29CD6C12  SSA Logical Disk Drive
  host1t:hdisk10                   29CD6D07  SSA Logical Disk Drive
  host1t:hdisk11                   29CD6D0A  SSA Logical Disk Drive
  host1t:hdisk12                   29CD6E19  SSA Logical Disk Drive
  host1t:hdisk13                   29CD6EDB  SSA Logical Disk Drive
  host1t:hdisk14                   29CD6EE1  SSA Logical Disk Drive
  host1t:hdisk15                   29CD70BD  SSA Logical Disk Drive
  host1t:hdisk16                   29CD70C5  SSA Logical Disk Drive
  host1t:hdisk17                   29CD731D  SSA Logical Disk Drive
  host1t:hdisk18                   29CD7372  SSA Logical Disk Drive
```

```
diag
   Task Selection(Diagnostics, Advanced Diagnostics, Service Aids, etc.)
      SSA Service Aids
         Link Verification
```

```
                host1t:ssa0 10-78   IBM SSA 160 SerialRAID Adapter (
```

```
Physical                              Serial#     Adapter Port
                                                    A1  A2  B1  B2      Status
  host1t:pdisk14                        29CD731D     0  10              Good
  host1t:pdisk11                        29CD6EE1     1   9              Good
  host1t:pdisk12                        29CD70BD     2   8              Good
  host1t:pdisk13                        29CD70C5     3   7              Good
  host1t:pdisk10                        29CD6EDB     4   6              Good
  host1t:pdisk9                         29CD6E19     5   5              Good
  host1t:pdisk1                         29CD697E     6   4              Good
  host1t:pdisk2                         29CD6980     7   3              Good
  host4t:ssa0:A                                      8   2
  host3t:ssa0:A                                      9   1
  host2t:ssa0:A                                     10   0
  host1t:pdisk15                        29CD7372                0  10      Good
  host1t:pdisk3                         29CD6A3B                1   9      Good
  host1t:pdisk5                         29CD6B0D                2   8      Good
  host1t:pdisk4                         29CD6ACA                3   7      Good
  host1t:pdisk7                         29CD6D07                4   6      Good
  host1t:pdisk0                         29CD5754                5   5      Good
  host1t:pdisk8                         29CD6D0A                6   4      Good
  host1t:pdisk6                         29CD6C12                7   3      Good
  host4t:ssa0:B                                                 8   2
  host3t:ssa0:B                                                 9   1
  host2t:ssa0:B                                                10   0
```

```
diag
   Task Selection(Diagnostics, Advanced Diagnostics, Service Aids, etc.)
      SSA Service Aids
         Physical Link Verification
             host1t:ssa0 10-78   IBM SSA 160 SerialRAID Adapter (
```

```
  Link   Port    Device                          Location      Port   Link
  40-I           host1t:pdisk14                  4C09-01        ||    40-C
  40-I           host1t:pdisk11                  4C09-02              40-I
  40-I           host1t:pdisk12                  4C09-03              40-I
  40-I    >>     host1t:pdisk13                  4C09-04              40-I
  40-I           host1t:pdisk10                  4C09-05        >>    40-I
  40-I           host1t:pdisk9                   4C09-06              40-I
```

```
40-I          host1t:pdisk1                    4C09-07          40-I
40-C   ||     host1t:pdisk2                    4C09-08          40-I
40-C   A2     host4t:ssa0                                 A1   40-C
40-C   A2     host3t:ssa0                                 A1   40-C
40-C   A2     host2t:ssa0                                 A1   40-C
40-C  A2     host1t:ssa0
              host1t:ssa0                                 B1   40-C
40-I          host1t:pdisk15                   4C09-09    ||   40-C
40-I          host1t:pdisk3                    4C09-10          40-I
40-I          host1t:pdisk5                    4C09-11          40-I
40-I   >>     host1t:pdisk4                    4C09-12          40-I
40-I          host1t:pdisk7                    4C09-13    >>    40-I
40-I          host1t:pdisk0                    4C09-14          40-I
40-I          host1t:pdisk8                    4C09-15          40-I
40-C   ||     host1t:pdisk6                    4C09-16          40-I
40-C   B2     host4t:ssa0                                 B1   40-C
40-C   B2     host3t:ssa0                                 B1   40-C
40-C   B2     host2t:ssa0                                 B1   40-C
40-C  B2     host1t:ssa0
```

B

Distributed software installation

After installing the base AIX operating system onto our four nodes using CDs and applying the program temporary fixes (PTFs), we had to choose the method to add HACMP and GPFS filesets to our systems. We chose the same method many SP administrators use, which is **`bffcreate`**. Instead of moving from node to node and inserting various CDs, we created an installable image of the software on an NFS mounted disk that all nodes could access. The added advantage of this method is once we know the **`installp`** syntax we want to use, we can send that command simultaneously to all nodes we want updated. We created the installable image on host1t which serves the NFS mounted filesystem /tools/images. All nodes in our cluster had to be able to communicate to each other and the NFS directory had with be executable from any node.

Creating the image

The simplest method is to run **smitty bffcreate** and follow the path to make an image of the desired software. Once in the smit screens, many filesets can be chosen to populate the images directory, but for ease of understanding we will demonstrate only sysmgt.websm. When we created update images, we put them in a separate directory. Due to naming conflicts, it is a bad idea to install from the same source that updates reside in.

```
host1t:/> smitty bffcreate

* INPUT device / directory for software                /dev/cd0
* SOFTWARE package to copy                            [sysmgt.websm          > +
* DIRECTORY for storing software package               [/tools/images]
  DIRECTORY for temporary storage during copying      [/tmp]
  EXTEND file systems if space needed?                 no                      +
  Process multiple volumes?                            yes                     +
```

Creating the installp command

We could now run **installp** from the command line, but we prefer to build the command using smit. A log is created whenever smitty runs. To simplify identifying the most recent activity, we removed smit.script and smit.log from /root. It is also important to have an updated .toc (table of contents) in the /tools/images directory, otherwise the **smitty install** command will miss filesets stored there. To update the .toc, run **rm .toc** then **inutoc .** from within /tools/images directory. We could choose to run **smitty -x install** which does not perform the install but does produce a usable smit.script command line. We chose to perform the install for the sake of testing on one node before propagating to all the other nodes.

```
host1t:/tools/images/> rm .toc
host1t:/tools/images/> inutoc .

host1t:/> rm smit.script
host1t:/> rm smit.log

host1t:/tools/images/> smitty install

->Install and Update Software
```

```
->Install and Update from ALL Available Software
->INPUT device / directory for software              [/tools/images]          +


* INPUT device / directory for software               /tools/images
* SOFTWARE to install                                [sysmgt.websm          > +
  PREVIEW only? (install operation will NOT occur)    no                      +
  COMMIT software updates?                            yes                     +
  SAVE replaced files?                                no                      +
  AUTOMATICALLY install requisite software?           yes                     +
  EXTEND file systems if space needed?                yes                     +
  OVERWRITE same or newer versions?                   no                      +
  VERIFY install and check file sizes?                no                      +
  DETAILED output?                                    no                      +
  Process multiple volumes?                           yes                     +
```

Propagating the fileset installation

The preceding SMIT produced the command **installp -acgNqwX -d /tools/images -f /tools/images/File 2>&1** after successfully installing our chosen fileset. We looked inside File and found bos.data, the only piece of software we wanted installed. Normally, File would be placed under /root and we would have to copy it to all the different nodes. In our case, using the NFS mounted directory created another simplification by making File immediately available to all nodes. At this point, we sent the **installp** command to all the rest of the nodes and then verified it was successful. Verification can be done through viewing the smit.log file on every node (the logs show detail about the installation) or by checking every node for the presence of the newly installed software.

```
gdsh -e host1t “installp -acgNqwX -d /tools/images -f /tools/images/File 2>&1”


host1t:/> gdsh -a "lslpp -l | grep sysmgt.websm.rte"
host1t:   sysmgt.websm.rte           4.3.3.0  COMMITTED  Web-based System
Manager
host2t:   sysmgt.websm.rte           4.3.3.0  COMMITTED  Web-based System
Manager
host3t:   sysmgt.websm.rte           4.3.3.0  COMMITTED  Web-based System
Manager
host4t:   sysmgt.websm.rte           4.3.3.0  COMMITTED  Web-based System
Manager
```

C

A useful tool for distributed commands

When working with multiple hosts, it is desirable to perform all system administration functions from one workstation. Administrators with an SP use **dsh** commands. We performed similar commands with **gdsh**, which actually uses the **rsh** command to perform its functions. To obtain a copy of the **gdsh** source code script, see Appendix H, “Additional material” on page 259 for details.

gdsh

This tool is available to download as described in Appendix H, “Additional material” on page 259. It is to be used “at your own risk”.

- **gdsh**
  - **-v** means verbose mode, extra output available during execution
  - **-a** is enable entire cluster
  - **-w** allows specific nodes to be targeted for commands
  - **-e** specifically excludes specific nodes from the stated command

We had to create a file under /root called all.nodes on every host machine which was a complete list of all the nodes we might want to select for remote command activity. We used the token ring addresses since we tried to reserve the ethernet for GPFS specific processes.

```
host1t:/> cat all.nodes
host1t
host2t
host3t
host4t

host1t:/> gdsh -w host4t "date"
host4t: Wed Feb 14 17:45:19 EST 2001

host1t:/> gdsh -e host4t,host3t "date"
host1t: Wed Feb 14 17:45:30 EST 2001
host2t: Wed Feb 14 17:43:44 EST 2001

host1t:/> gdsh -a "date"
host1t: Wed Feb 14 17:45:59 EST 2001
host2t: Wed Feb 14 17:44:13 EST 2001
host3t: Wed Feb 14 17:44:21 EST 2001
host4t: Wed Feb 14 17:46:41 EST 2001
```

```
#!/usr/bin/perl
#
#
use Getopt::Std;

getopts("vae:w:", \%option);

if ($option{v} ) {
```

```
    print "Verbose mode enabled\n";
    if ($option{a} ) {
        print "Entire cluster mode enabled\n";
    }
    if ($option{e} ) {
        print "some nodes are being excluded\n";
    }
    if ($option{w} ) {
        print "specific nodes are being included\n";
    }
print "\n";
}

$node_ctr=-1;

if ($option{e} ) {
    @exclude_nodes=split(/\,/, $option{e} );
    foreach $t (@exclude_nodes) {
        $excluded_nodes{$t} = "yes";
        if ($option{v} ) {
            print "Excluding $t\n";
        }
    }
}

if (! ( $option{a} || $option{w}) ) {

    if ( -r "/all.nodes" )  {
        open (NODELIST, "< /all.nodes");
        while ( <NODELIST> ) {
            chomp;
            if ( index($_,"#") != 0 ) {
                if ( ! defined $excluded_nodes{$_} ) {
                    $node_ctr++;
                    $node_name[$node_ctr] = $_;
                }
            }
         }
        close (NODELIST);
    } else {
        print "The node list file /all.nodes was not found. Exiting.\n";
        exit(1);
      }

}  elsif ($option{w} ) {   # option w
    @include_nodes=split(/\,/, $option{w} );
    foreach $node (@include_nodes) {
        $node_ctr++;
        $node_name[$node_ctr] = $node;
```

```
    }


}  else {   # option a is only one left
    open (NODELIST, "/usr/sbin/cluster/utilities/clhandle -a |");
    while ( <NODELIST> ) {
        chomp;
        @f=split;
        if ( ! defined $excluded_nodes{$f[1]} ) {
            $node_ctr++;
            $node_name[$node_ctr] = $f[1];
        }
    }
    close (NODELIST);
}


if ($option{v} ) {
    print "Current working collective:\n";
    foreach $node (@node_name) {
        print "$node\n";
    }
$i=$#ARGV +1;
print "Number parms is: $i \n";
}

if ( $#ARGV >=0 ) {
    $i=0;

    while ( $i <= $#ARGV ) {
        $parm_string = $parm_string . " " . $ARGV[$i] ;
        $i++;
    }
    if ($option{v} ) {
        print "The complete parm string is: $parm_string\n";
    }
} else {
    print "Usage: gdsh -ave node1,node2 cmds \n";
    print "  where:\n";
    print "    -a  -> all nodes in hacmp cluster (as reported by clhandle
-a)\n";
    print "    -v  -> verbose mode\n";
    print "    -e  -> nodes to exclude in comma separated list\n";
    print "    -w  -> nodes to include in comma separated list\n";
    print " Command will use /all.nodes as working collective\n";
    exit(1);
}
```

```
foreach $node (@node_name) {
    $t = `rsh $node $parm_string 2>&1`;

    if ($t ne "") {
        @f = split (/$\//, $t);

        foreach $line (@f) {
            print "$node: $line\n";
        }
    }
}
```

D

Useful scripts

This appendix describes some useful scripts that you may use “at your own risk.” These scripts are provided as suggestions and not the only way to perform any specific action. To obtain a copy of the source code of any of these scripts, see Appendix H, “Additional material” on page 259 for details.

Creating GPFS disks

The following three scripts provide an alternate automated technique to define volume groups and logical volumes for use by GPFS.

mkvgdriver This script will create a "driver" file for the next script. This is the script where we describe the disks to be used by GPFS and provide the volume group and logical volume naming standards.

mkgpfsvg This script will create the actual volume group and logical volume on the disk and set the "AUTO ON" volume group property. After this script has completed all the volume groups and logical volumes will be known to one of the nodes in the cluster.

importvgs.to.newtail This script will be run from the node where the volume groups and logical volumes have been defined. Its function will be to import these LVM disk definitions to the other nodes in the cluster.

These scripts follow the same principles described in Chapter 6, “Configuring GPFS and SSA disks” on page 91 but provide some ideas on how this task can be further automated.

The same process is followed using these three scripts:

- The volume groups and logical volumes are defined completely on one node in the cluster.
- The volume groups are then imported to all the other nodes in the cluster.

These scripts will be described in some detail and an example will be given. As these scripts are general in nature, they will have to be modified for each use.

The first script, mkvgdriver, will output a series of commands that will in turn call the mkgpfsvg script. The disks that will be used are coded in this first script.

Example: D-1 mkvgdriver script

```
#!/usr/bin/ksh
#
# script will create a driver file for the mkgpfsvg script.
# this will list all SSA disks that have successfully configured
# hdisks and create a mkgpfsvg command for that disk.
#
vg_prefix="gpfsvg"
lv_prefix="gpfslv"
#
start_count=0  # where to start the volume group count from
```

```
#
(( disk_count = $start_count ))
#
# get all Available pdisk names
for pdisk in $( lsdev -Ccpdisk -S Available -F name )
do
    hdisk=$( ssaxlate -l $pdisk ) # get the hdisk to be sure its configured
    if [[ -n $hdisk ]] then
        vgname=$( lspv|grep -w $hdisk | awk '{print $3}' ) # not other vg
        if [[ $vgname = "None" ]] then
            echo mkgpfsvg $hdisk ${vg_prefix}${disk_count}
${lv_prefix}${disk_count}
            (( disk_count += 1 ))
        else
            echo Disk $hdisk belongs to vg $vgname.  Skipping
        fi
    else
        echo Error, no logical hdisk for pdisk $pdisk.  Skipping disk.
    fi
done
```

In this example of mkvgdriver, all the SSA disks will be used. The volume groups will be called gpfsvgXX and the logical volumes will be called gpfslvXX, where XX is a counter starting at zero and incriminating by one for each disk.

This script can be customized in many ways. For instance, if only the SSA disks in a specific enclosure were to be included, the **`lsdev -Ccpdisk -S Available -F name`** could be changed to list only them.

To only use the SSA disks in enclosure 30-58-P, the command would be:

```
(00:37:52) c185n01:/ # lsdev -Ccpdisk|grep 30-58-P
pdisk0  Available 30-58-P 4GB SSA C Physical Disk Drive
pdisk1  Available 30-58-P 4GB SSA C Physical Disk Drive
pdisk2  Available 30-58-P 4GB SSA C Physical Disk Drive
pdisk3  Available 30-58-P 4GB SSA C Physical Disk Drive
pdisk4  Available 30-58-P 4GB SSA C Physical Disk Drive
pdisk5  Available 30-58-P 4GB SSA C Physical Disk Drive
pdisk6  Available 30-58-P 4GB SSA C Physical Disk Drive
pdisk7  Available 30-58-P 4GB SSA C Physical Disk Drive
pdisk8  Available 30-58-P 4GB SSA C Physical Disk Drive
pdisk9  Available 30-58-P 4GB SSA C Physical Disk Drive
pdisk10 Available 30-58-P 4GB SSA C Physical Disk Drive
pdisk11 Available 30-58-P 4GB SSA C Physical Disk Drive
pdisk12 Available 30-58-P 4GB SSA C Physical Disk Drive
pdisk13 Available 30-58-P 4GB SSA C Physical Disk Drive
pdisk14 Available 30-58-P 4GB SSA C Physical Disk Drive
pdisk15 Available 30-58-P 4GB SSA C Physical Disk Drive
```

```
(00:37:55) c185n01:/ # lsdev -Ccpdisk|grep 30-58-P|awk  '{print $1}'
pdisk0
pdisk1
pdisk2
pdisk3
pdisk4
pdisk5
pdisk6
pdisk7
pdisk8
pdisk9
pdisk10
pdisk11
pdisk12
pdisk13
pdisk14
pdisk15
```

By piping the **lsdev** command into different **grep** commands this script can be very flexible.

The output of the mkvgdriver script will be a list of **mkgpfsvg** commands. These should be saved in a file to be run in the next step.

```
(00:41:52) c185n01:/u/gmcpheet/hacmp/scripts # running mkvgdriver
mkgpfsvg hdisk2 gpfsvg2 gpfslv2
mkgpfsvg hdisk3 gpfsvg3 gpfslv3
mkgpfsvg hdisk4 gpfsvg4 gpfslv4
mkgpfsvg hdisk5 gpfsvg5 gpfslv5
mkgpfsvg hdisk6 gpfsvg6 gpfslv6
mkgpfsvg hdisk7 gpfsvg7 gpfslv7
mkgpfsvg hdisk8 gpfsvg8 gpfslv8
mkgpfsvg hdisk9 gpfsvg9 gpfslv9
mkgpfsvg hdisk10 gpfsvg10 gpfslv10
mkgpfsvg hdisk11 gpfsvg11 gpfslv11
mkgpfsvg hdisk12 gpfsvg12 gpfslv12
mkgpfsvg hdisk13 gpfsvg13 gpfslv13
mkgpfsvg hdisk14 gpfsvg14 gpfslv14
mkgpfsvg hdisk15 gpfsvg15 gpfslv15
mkgpfsvg hdisk16 gpfsvg16 gpfslv16
mkgpfsvg hdisk17 gpfsvg17 gpfslv17
```

To save this in a script: `mkvgdriver > mkvgdriver.output`

On to the next step, the actual creation of volume groups and logical volumes.

The mkgpfsvg script is the script that creates the volume groups and logical volumes.

Example: D-2 mkgpfsvg script

```
#!/usr/bin/ksh
#
#
# script to assist with the definitions of gpfsvg and lvs
#
#
#
# this is the proper set up for 4.5 GB SSA disks
#
size_pps=16    # pp size
#
#num_pps=542   #9.1 GB at 16MB
num_pps=268    #4.5GB  at 16MB
#
if [[ $# -ne 3 ]] then
    echo Incorrect number of parms.
    echo Usage is $0 hdisk vgname lvname
    exit 1
fi
#
hdisk=$1
vgname=$2
lvname=$3
#
echo
echo Now creating a volume group $vgname with lv $lvname on hdisk $hdisk
#
# verify passed disk is SSA
#
pdisk=$(ssaxlate -l $hdisk)
rc=$?
if [[ $rc != 0 ]] then
    echo hdisk $hdisk is not a SSA disk.
    exit 1
fi
#
# verify vg does not already exist
#
vgcount=$( lspv | grep -w $vgname | wc -l )
if [[ $vgcount -ne 0 ]] then
    echo A volume group with name already exists.
    exit 1
fi
#
# verify lv does not already exist
```

```
#
lvcount=$( /usr/sbin/getlvodm -l $lvname > /dev/null 2>&1 )
rc=$?
if [[ $rc = 0 ]] then
    echo There is a LV with the name $lvname.
    exit 1
fi
#
# Issue the mkvg command
#
mkvg -f -n -s $size_pps -c -y $vgname $hdisk
rc=$?
if [[ $rc != 0 ]] then
    echo Error creating vg $vgname on $hdisk. rc=$rc
    exit 1
fi
#
varyonvg $vgname
rc=$?
if [[ $rc != 0 ]] then
    echo Error varying on vg $vgname.  rc=$rc
    exit 1
fi
#
mklv -b n -w n -t mmfsha -x $num_pps -y $lvname $vgname $num_pps
rc=$?
if [[ $rc != 0 ]] then
    echo Error making lv $lvname on vg $vgname.  rc=$rc
    exit 1
else
    echo Successfully created lv $lvname on $vgname
fi
#
varyoffvg $vgname
rc=$?
if [[ $rc != 0 ]] then
    echo Error varying off vg $vgname.  rc=$rc
    exit 1
fi
#
echo Successfully completed creation of $vgname containing $lvname on disk
$hdisk
```

This script runs the commands which are the output from the mkvgdriver script. This script is also very flexible and this example demonstrates this. In this script the PP size and number of PPs to fit on one SSA disk must be coded. These are the parameters num_pps and size_pps.

This script will then confirm that:

1. The disk being used is an SAA disk
2. The volume group name being requested doesn't exist
3. The logical volume name being requested doesn't exist

After these tests are passed, the **mkvg** command is issued, the volume group is then varied online and the **mklv** command is issued. After the **mklv** has completed the disk is varied offline.

After this script completes all the disks selected in the mkvgdriver script will have volume groups and logical volumes on them in the proper state for use by GPFS.

The next step is to import these volume groups to all the other nodes in the cluster.

The importvgs.to.newtail script serves this purpose.

Example: D-3 importvgs.to.newtail script

```
#!/usr/bin/ksh
#
# this script will import volume groups on a multi tailed SSA config
# from a node where they are defined to a node in the loop where
# the vgs are not known.
#
# The script needs to be run on the node where the vgs are
# currently defined and the "remote_node" is the node where they
# need to be imported.
#
remote_node=$1
if [[ -z $remote_node ]] then
    echo Usage:  importvgs.to.newtail nodename
    echo "    where nodename is the node to import the vgs to."
    echo "    The node where this command is run must already have the vgs
imported."
    exit 1
fi
#
lspv|grep gpfs|\
while read disk pvid vg
do
remote_vg=$(rsh $remote_node -n "lspv|grep $pvid")
remote_vgname=$(echo $remote_vg|awk '{print $3}')
remote_hdisk=$(echo $remote_vg|awk '{print $1}')
#
if [[ "$vg" = "$remote_vgname" ]] then
    echo $disk which is pvid $pvid vg $vg is known as vg $remote_vgname on
$remote_node. OK
```

```
else
    echo $disk which is pvid $pvid vg $vg is known as vg $remote_vgname on
$remote_hdisk on $remote_node.
    if [[ $remote_vgname = "None" ]] then
        echo Running importvg -y $vg $remote_hdisk on $remote_node
        rsh $remote_node -n importvg -y $vg $remote_hdisk
        echo chvg -a n $vg
        rsh $remote_node -n chvg -a n $vg
        echo varyoffvg $vg
        rsh $remote_node -n varyoffvg $vg
    else
        echo $disk pvid $pvid vg $remote_vgname on $remote_node has a volume
group.  Not processed.
    fi
fi
#
done
```

This script is run on the node where all of the volume groups have already been defined. This script expects one parameter, that of the node where the disks are to be imported.

For example, if the node c185n08 has all the SSA disks defined, and node c185n01 needs them imported, then the command is issued on node c185n08 with the parameter of c185n01.

Before the command is issued, the disks should look like this on node c185n01:

```
hdisk2          000167498707c169                    gpfsvg16
hdisk3          00001351566acb07                    gpfsvg17
hdisk4          00001351566ae6aa                    gpfsvg18
hdisk5          00001351566b0f35                    gpfsvg19
hdisk6          0000099017a37f8c                    gpfsvg20
hdisk7          00001351566b378d                    gpfsvg21
hdisk8          00001351566b4520                    gpfsvg22
hdisk9          00001351566b52f3                    gpfsvg23
hdisk10         00001351566b60a7                    gpfsvg24
hdisk11         00001351566b6e63                    gpfsvg25
hdisk12         00001351566b7be6                    gpfsvg26
hdisk13         0002025690cec645                    gpfsvg27
hdisk14         00001351566b8977                    gpfsvg28
hdisk15         00001351566c1e3d                    gpfsvg29
hdisk16         00001351566c2bea                    gpfsvg30
hdisk17         00001351566c6231                    gpfsvg31
```

And the node to be imported to c185n01 should look like this:

```
hdisk2          000167498707c169                    None
hdisk3          00001351566acb07                    None
```

```
hdisk4          00001351566ae6aa                    None
hdisk5          00001351566b0f35                    None
hdisk6          0000099017a37f8c                    None
hdisk7          00001351566b378d                    None
hdisk8          00001351566b4520                    None
hdisk9          00001351566b52f3                    None
hdisk10         00001351566b60a7                    None
hdisk11         00001351566b6e63                    None
hdisk12         00001351566b7be6                    None
hdisk13         0002025690cec645                    None
hdisk14         00001351566b8977                    None
hdisk15         00001351566c1e3d                    None
hdisk16         00001351566c2bea                    None
hdisk17         00001351566c6231                    None

(01:06:44) c185n08:/u/gmcpheet/hacmp/scripts # importvgs.to.newtail c185n01
hdisk2 which is pvid 000167498707c169 vg gpfsvg16 is known as vg None on hdisk2
on c185n01.
Running importvg -y gpfsvg16 hdisk2 on c185n01
gpfsvg16
0516-783 importvg: This imported volume group is concurrent capable.
        Therefore, the volume group must be varied on manually.
chvg -a n gpfsvg16
0516-1260 chvg: Device configuration database has been updated with new
information.
       Since the volume group gpfsvg16 is not varied on, if the gpfsvg16 is of
Big
        Volume group type, chvg command must be run with the volume group
varied
        on for these attributes to be saved across exportvg/importvg operation.
varyoffvg gpfsvg16
0516-010 lvaryoffvg: Volume group must be varied on; use varyonvg command.
0516-942 varyoffvg: Unable to vary off volume group gpfsvg16.
hdisk3 which is pvid 00001351566acb07 vg gpfsvg17 is known as vg None on hdisk3
on c185n01.
Running importvg -y gpfsvg17 hdisk3 on c185n01
gpfsvg17
0516-783 importvg: This imported volume group is concurrent capable.
        Therefore, the volume group must be varied on manually.
chvg -a n gpfsvg17
0516-1260 chvg: Device configuration database has been updated with new
information.
       Since the volume group gpfsvg17 is not varied on, if the gpfsvg17 is of
Big
        Volume group type, chvg command must be run with the volume group
varied
        on for these attributes to be saved across exportvg/importvg operation.
varyoffvg gpfsvg17
0516-010 lvaryoffvg: Volume group must be varied on; use varyonvg command.
```

```
0516-942 varyoffvg: Unable to vary off volume group gpfsvg17.
hdisk4 which is pvid 00001351566ae6aa vg gpfsvg18 is known as vg None on hdisk4
on c185n01.
Running importvg -y gpfsvg18 hdisk4 on c185n01
```

..... this command produces more output then is shown here.

This output, although verbose, is actually successful. The **chvg** command is issuing a warning stating that this command was issued with the volume group varied offline. It then warns that if this volume group is of the type "Big Volume," more action is required. However, these volume groups are not of this type therefore no further action is required.

After this script runs the disks on node c185n01:

```
hdisk2          000167498707c169                gpfsvg16
hdisk3          00001351566acb07                gpfsvg17
hdisk4          00001351566ae6aa                gpfsvg18
hdisk5          00001351566b0f35                gpfsvg19
hdisk6          0000099017a37f8c                gpfsvg20
hdisk7          00001351566b378d                gpfsvg21
hdisk8          00001351566b4520                gpfsvg22
hdisk9          00001351566b52f3                gpfsvg23
hdisk10         00001351566b60a7                gpfsvg24
hdisk11         00001351566b6e63                gpfsvg25
hdisk12         00001351566b7be6                gpfsvg26
hdisk13         0002025690cec645                gpfsvg27
hdisk14         00001351566b8977                gpfsvg28
hdisk15         00001351566c1e3d                gpfsvg29
hdisk16         00001351566c2bea                gpfsvg30
hdisk17         00001351566c6231                gpfsvg31
```

This script would then be called for all the nodes in the cluster, for example:

```
importvgs.to.newtail c185n01
importvgs.to.newtail c185n02
importvgs.to.newtail c185n03
importvgs.to.newtail c185n04
```

This would import the volume group definitions known on the node where it is run to nodes c185n01/c185n02/c185n03 and c185n04.

comppvid

This script is used to compare the volume groups to the PVIDs on each node to make sure they are in sync. This assures the administrator that each PVID is known on each node by only one volume group name.

```
#!/usr/bin/ksh
# a script to compare what volume groups were created on each node
#
for i in `lspv |grep -v rootvg | awk '{print $2}'`
do
gdsh -a "lspv | grep $i"
echo "**************************************"
done


host1t:/tools/ralph> comppvid
host1t: hdisk2          000b4a7d1075fdbf    toolsvg
**************************************
host1t: hdisk3          000007024db58359    gpfsvg0
host2t: hdisk3          000007024db58359    gpfsvg0
host3t: hdisk2          000007024db58359    gpfsvg0
host4t: hdisk2          000007024db58359    gpfsvg0
**************************************
host1t: hdisk4          000007024db5472e    gpfsvg1
host2t: hdisk4          000007024db5472e    gpfsvg1
host3t: hdisk3          000007024db5472e    gpfsvg1
host4t: hdisk3          000007024db5472e    gpfsvg1
**************************************
host1t: hdisk5          000007024db54fb4    gpfsvg2
host2t: hdisk5          000007024db54fb4    gpfsvg2
host3t: hdisk4          000007024db54fb4    gpfsvg2
host4t: hdisk4          000007024db54fb4    gpfsvg2
**************************************
host1t: hdisk6          000007024db5608a    gpfsvg3
host2t: hdisk6          000007024db5608a    gpfsvg3
host3t: hdisk5          000007024db5608a    gpfsvg3
host4t: hdisk5          000007024db5608a    gpfsvg3
**************************************
host1t: hdisk7          000007024db571ba    gpfsvg4
host2t: hdisk7          000007024db571ba    gpfsvg4
host3t: hdisk6          000007024db571ba    gpfsvg4
host4t: hdisk6          000007024db571ba    gpfsvg4
**************************************
host1t: hdisk8          000007024db5692c    gpfsvg5
host2t: hdisk8          000007024db5692c    gpfsvg5
host3t: hdisk7          000007024db5692c    gpfsvg5
host4t: hdisk7          000007024db5692c    gpfsvg5
**************************************
host1t: hdisk9          000158511eb0f296    gpfsvg6
host2t: hdisk9          000158511eb0f296    gpfsvg6
host3t: hdisk8          000158511eb0f296    gpfsvg6
host4t: hdisk8          000158511eb0f296    gpfsvg6
```

```
**************************************
host1t: hdisk10        000007024db57a49    gpfsvg7
host2t: hdisk10        000007024db57a49    gpfsvg7
host3t: hdisk9         000007024db57a49    gpfsvg7
host4t: hdisk9         000007024db57a49    gpfsvg7
**************************************
host1t: hdisk11        000007024db58bd3    gpfsvg8
host2t: hdisk11        000007024db58bd3    gpfsvg8
host3t: hdisk10        000007024db58bd3    gpfsvg8
host4t: hdisk10        000007024db58bd3    gpfsvg8
**************************************
host1t: hdisk12        000007024db53eac    gpfsvg9
host2t: hdisk12        000007024db53eac    gpfsvg9
host3t: hdisk11        000007024db53eac    gpfsvg9
host4t: hdisk11        000007024db53eac    gpfsvg9
**************************************
host1t: hdisk13        000007024db5361d    gpfsvg10
host2t: hdisk13        000007024db5361d    gpfsvg10
host3t: hdisk12        000007024db5361d    gpfsvg10
host4t: hdisk12        000007024db5361d    gpfsvg10
**************************************
host1t: hdisk14        000007024db51c4b    gpfsvg11
host2t: hdisk14        000007024db51c4b    gpfsvg11
host3t: hdisk13        000007024db51c4b    gpfsvg11
host4t: hdisk13        000007024db51c4b    gpfsvg11
**************************************
host1t: hdisk15        000007024db524ce    gpfsvg12
host2t: hdisk15        000007024db524ce    gpfsvg12
host3t: hdisk14        000007024db524ce    gpfsvg12
host4t: hdisk14        000007024db524ce    gpfsvg12
**************************************
host1t: hdisk16        000007024db52d7b    gpfsvg13
host2t: hdisk16        000007024db52d7b    gpfsvg13
host3t: hdisk15        000007024db52d7b    gpfsvg13
host4t: hdisk15        000007024db52d7b    gpfsvg13
**************************************
host1t: hdisk17        000007024db513d2    gpfsvg14
host2t: hdisk17        000007024db513d2    gpfsvg14
host3t: hdisk16        000007024db513d2    gpfsvg14
host4t: hdisk16        000007024db513d2    gpfsvg14
**************************************
host1t: hdisk18        000007024db55810    gpfsvg15
host2t: hdisk18        000007024db55810    gpfsvg15
host3t: hdisk17        000007024db55810    gpfsvg15
host4t: hdisk17        000007024db55810    gpfsvg15
**************************************
```

E

Subsystems and Log files

This appendix gives a listing of subsystems of RSCT and HACMP/ES, and log files.

Subsystems of HACMP/ES

The following is a listing of the subsystems of HACMP/ES. The corresponding name in an SP environment appears in brackets.

Name	Subsystem	Group	Daemon
Topology Services (High Availability Topology Services, HATS)	topsvcs (hats)	topsvcs	/usr/sbin/rsct/bin/hatsd
Group Services (High Availability Group Services, HAGS)	grpsvcs (hags)	grpsvcs	/usr/sbin/rsct/bin/hagsd
Cluster Globalized Server Daemon	grpglsm	grpsvcs	/usr/sbin/rsct/bin/hagsglsmd
Event Management (High Availability Event Management, HAEM)	emsvcs	emsvcs	/usr/sbin/rsct/bin/haemd
Event Management AIX Operating System Resource Monitor	emaixos	emsvcs	/usr/sbin/rsct/bin/emaixos
HACMP/ES Cluster Manager	clstrmgrES	cluster	/usr/es/sbin/cluster/clstrmgrES
HACMP/ES Cluster SMUX Peer deamon	clsmuxpdES	cluster	/usr/es/sbin/cluster/clsmuxpdES
HACMP/ES Cluster Information Program daemon	clinfoES	cluster	/usr/es/sbin/cluster/clinfoES
HACMP/ES Cluster Lock Manager daemon	cllockdES	lock	/usr/es/sbin/cluster/cllockdES

Log files for the RSCT component

Trace files

/var/ha/log contains the trace files for daemons.

Topology Services

topsvcs.*DD.HHMMSS.name* in a non SP environment
hats.*DD.HHMMSS.name* in a SP environment

DD	day of the month
HHMMSS	hours, minutes, seconds of the day
name	HACMP/ES cluster name

Group Services

grpsvcs_*nodenum_instnum.name* in a non SP environement
hats_*nodenum_instnum.name* in a SP environment

nodenum	node number, as determined by clhandle, ...
instnum	instance number of daemon
name	HACMP/ES cluster name

Working directories

/var/ha/run contains a directory for each domain of a RSCT subsystem. Each directory contains the core files (find out how many here)

In a non SP environment those are

topsvcs.*name*
grpsvcs.*name*
emsvcs.*name*

name is the HACMP/ES cluster name.

In an SP environment those are

hats.*syspar_nam*e
hags.*syspar_name*
haem.*syspar_name*

syspar.name is the name of the system partition

Log files for the cluster group

clstrmgrES

/tmp contains log files that contain debug information for the cluster manager daemon.

clstrmgr.debug
clstrmgr.debug.1
clstrmgr.debug.bak
clstrmgr.debug.bak.1

clresmgrd.

When the cluster manager is started, the file clstrmgr.debug is moved to clstrmgr.debug.1. In during the runtime of the cluster manager the log file exceeds the length of .. , ...

clstrmgr.debug.bak and clstrmgr.debug.bak.1 are the files of the last two ...

clsmuxpdES

/usr/es/adm/cluster.log

clinfoES

/tmp contains log files

clinfo.rc.out
clinfo.rc.out.*n*

where *n* is the file

Log files generated by HACMP/ES utilities

Event history

/tmp/hamcp.out

/usr/es/sbin/cluster/history/cluster.*mmdd*

/tmp/cspoc.log

F

Summary of commands

The following is a compilation of some of the commands we used while working with HACMP and GPFS. We created this list to serve the wide range of experience (or lack of) for the folks managing GPFS. This list is not definitive and we suggest that you refer to the man pages to understand the syntax required for each one.

GPFS commands

mmaddcluster	Adds nodes to an existing GPFS cluster
mmadddisk	Adds disks to a GPFS file system
mmaddnode	Adds nodes to a GPFS nodeset
mmchattr	Changes attributes of one or more GPFS files
mmchcluster	Changes the primary or secondary GPFS SDR server
mmchconfig	Changes GPFS configuration attributes
mmchdisk	Changes state or parameters of one or more disks
mmcheckquota	Checks file system i-node and space usage
mmchfs	Changes the attributes of a GPFS file system
mmconfig	Defines and configures a new GPFS nodeset
mmcrcluster	Create a GPFS cluster from a set of nodes
mmcrfs	Creates a GPFS file system
mmdefragfs	Reduces disk fragmentation by increasing the number of full free blocks available to the file system.
mmdelcluster	Deletes nodes from an existing GPFS cluster
mmdeldisk	Removes a disk from a GPFS file system
mmdelfs	Removes a GPFS file system
mmdelnode	Removes one or more nodes from a GPFS nodeset
mmfsadm cleanup	Attempts to repair mmfs subsystems
mmfsadm dump cfgmgr	Reveals detailed about mmfs configuration
mmfsadm dump waiters	Used for mmfs problem determination
mmfsadm shutdown	Orderly stop of mmfs subsystems
mmfsck	Checks and repairs a GPFS file system
mmlsattr	Queries file attributes
mmlscluster	Displays GPFS cluster information
mmlsdisk	Displays configuration and state of disks in a file system
mmlsquota	Displays quota information for a user or group
mmquotaoff	Deactivates quota limit checking
mmquotaon	Activates quota limit checking
mmrestripefs	Rebalances or restores the replication factor of all files

mmrpldisk	Replaces the specified disk
mmshow_fence	Displays nodes fenced out by disk
mmstartup	Starts GPFS on some or all nodes
mmshutdown	Unmounts all GPFS file systems and stops GPFS on one or more nodes

SSA commands

ssa_speed	Displays speed of SSA loops
ssaadap	Lists the adapters to which an SSA logical disk or physical disk is connected
ssacand	Displays the unused connection locations for an SSA adapter
ssaconn	Displays the SSA connection details for the physical disk
ssadisk	Returns all pdisks or all hdisks attached to a node
ssaidentify	Blinks the light on a specific SSA disk
ssaxlate	Translates pdisk to hdisk and hdisk to pdisk

AIX commands

bffcreate	Creates installation image files in backup format
cfgmgr	Configures devices and optionally installs device software by running the programs specified in the Configuration Rules object class.
exportvg	Exports the definition of a volume group from a set of physical volumes
filemon	Monitors the performance of the file system, and reports the I/O activity on behalf of logical files, virtual memory segments, logical volumes, and physical volumes
importvg	Imports a new volume group definition from a set of physical volumes
installp	Installs available software products in a compatible installation package
iostat	Reports Central Processing Unit (CPU) statistics and input/output statistics for tty devices, disks, and CD-ROMS.

lsattr	Displays attribute characteristics and possible values of attributes for devices in the system.
lscfg	Displays system configuration data
lsdev -Cc disk	Displays disks in the system and their characteristics
lslpp	Lists software products
lslv	Displays information about a logical volume
lssrc	Gets the status of a subsystem
mklv	Creates a logical volume
mkvg	Creates a volume group
netstat	Shows network status
odmget	Retrieves objects from the specified object classes into an odmadd input file
oslevel	Reports the latest installed maintenance level of the system
rmdev	Unconfigures and undefines specified devices
setclock	Sets the time and date for a host on a network
varyoffvg	Deactivates a volume group
varyonvg	Activates a volume group
vmstat	Reports virtual memory statistics

HACMP commands

claddnode	Adds adapter entry to the adapter ODM object class HACMPadapter
cldare	Updates the HACMP Cluster Manager and associated daemons with new
clhandle	Gives node handle
clstat	Monitors the status of an HACMP cluster
clstop	Stops the Cluster daemons
rc.cluster	Starts the cluster services subsystem on a node, all subsystems on which it depends, and a specified set of clients

G

Benchmark and Example Code

This appendix contains descriptions of the benchmark programs used in this book and a complete source listing with additional commentary for GMGH (which is carefully documented to augment its role as an example). Source code (in electronic form) for the benchmark programs used in this book, plus other related programs are also available; see Appendix H, “Additional material” on page 259 for details.

In particular, this appendix contains:

- A summary of the benchmark programs used in this book
- Source listing plus highlights for GMGH

The benchmark programs

This section summarizes the benchmark programs and discusses how to use them. It also discusses matters related to linking them.

Summary of the benchmark programs

Four programs are used to generate the benchmark results sited in Chapter 8, “Developing Application Programs that use GPFS” on page 143. These programs also provide an example of how to use GPFS. They are available electronically (see Appendix H, “Additional material” on page 259 for more information). These four programs are:

ibm_sgw Write to a GPFS file using different I/O access patterns. If a random pattern is chosen, GPFS hints are not used.

ibm_sgr Read from a GPFS file using different I/O access patterns. If a random pattern is chosen, GPFS hints are not used.

ibm_shw Write to a GPFS file using the random I/O access pattern only. GPFS hints are used.

ibm_shr Read from a GPFS file using the random I/O access pattern only. GPFS hints are used.

Using the benchmark programs

Each program can be executed as a command with parameters. The parameters are the same for each program, but in some cases there are restrictions on the parameters. The syntax is: <prog_name> <path_file> <rec_size> <num_rec> <crunch> <order> <stride>

<prog_name> Program name

<path_file> This is the name of the file being accessed. It must include the full path, otherwise it will access a file in the same directory where the program is being executed. The path implicitly specifies the file system is being used; specifying a GPFS directory results in using GPFS. ibm_sgw and ibm_sgr can access non-GPFS file systems, but the other programs can not. Also note that if the file already exists, ibm_sgw and ibm_shw will overwrite it.

<rec_size> Record size given in bytes

<num_rec> This is the number of records being accessed. If the file is being written, <rec_size> * <num_rec> is the file size.

<crunch> Specifies whether or not the program includes significant amounts of CPU activity (i.e., number crunching) other than that needed strictly for doing I/O. There are two options:
yes - Do some number crunching to simulate a balanced CPU and I/O load. The number crunching loop may need to be tuned to the CPU model to keep the load balanced.
no - Do *no* number crunching.

<order> Specifies the I/O access pattern. Section 8.4, “Analysis of I/O access patterns” on page 154 discusses several of these patterns. Five case *in*sensitive options are possible:
seq - sequential
strd - strided
rndm - random
bkwd - backward sequential
bkst - backward strided

<stride> Specifies the stride for the <order> options strd and bkst.

The following example illustrates the use of one of these programs. It specifies no for the number crunching option. By taking the difference between the I/O time and overall time, CPU time devoted to non-I/O tasks can be measured implicitly; with number crunching turned off, the processing time is negligible.

```
host1t:/> ibm_sgw /gpfs1/L.strd 1048576 5120 no strd 4
----------------------------------------
JOB:  ibm_sgw
User Parameter Summary
----------------------------------------
base file name            = /gpfs1/L.strd
buffer size               = 1048576
number of records         = 5120
simulate number crunching = no
processing order          = strd
stride                    = 4
----------------------------------------
summary statistics
----------------------------------------
data processed     =  5120.0 MB
I/O time           =    77.573 sec
overall time       =    77.637 sec
Amortized I/O rate =    65.948 MB/s
----------------------------------------
```

The following example illustrates the use of another one of these programs. It specifies yes for the number crunching option. In this case, CPU time devoted to non-I/O activities is explicitly measured as crunch time. By tuning the number crunching loop, the ratio of I/O time to crunch time can be altered.

```
host1t:/> ibm_shr /gpfs1/Mc.rndm 262144 20480 yes rndm
--------------------------------
JOB:  ibm_shr
User Parameter Summary
--------------------------------
base file name            = /gpfs1/Mc.rndm
buffer size               = 262144
number of records         = 20480
simulate number crunching = yes
processing order          = rndm
----------------------------------------
summary statistics
----------------------------------------
data processed     =  5120.0 MB
crunch time        =    98.748 sec
I/O time           =    87.532 sec
overall time       =   186.670 sec
Amortized I/O rate =    27.428 MB/s
----------------------------------------
```

Linking the benchmark programs

Depending on the configuration of the system you use to make these benchmark codes, it may be desired (or even necessary) to make some modifications.

First, the benchmark codes use a random number generator to generate seek offsets for the random I/O access patterns. The random generator used in these programs is good, but if the reader wishes, it can be replaced with other ones. In particular, it has been coded to use the same API as the one provided in the ESSL library, surand(). If the ESSL version is preferred, the makefile can be modified by removing all references to ibm_urd and adding the -lessl flag to the xlc line. The programs will then work without modification to the source code.

Second, the benchmark programs use rtc() as a timer to calculate I/O rates. rtc() is provided in the Fortran 90 library libxlf90. It is used because of its very low overhead and high granularity compared with other timers. If this library is not available on the reader's system, then the rtc() system call can be replaced with

the gettimeofday() system call and the -lxlf90 flag may be removed from the makefile xlc lines. By comparing Example G-1and Example G-2 the reader can easily see how to replace rtc() with gettimeofday() in the benchmark codes, if necessary.

Example: G-1 Using rtc()

```
#include <stdio.h>
#include <time.h>

double rtc();

int main()
{
   int    k;
   double xsum = 0.0;
   double bg, dn, delta;

   bg = rtc();
   for (k = 0; k < 10000000; k++)
      xsum += (double)k;
   dn = rtc();
   delta = dn - bg;

   printf("delta = %10.6lf sec\n", delta);

   return 0;
}
```

Example: G-2 Using gettimeofday()

```
#include <stdio.h>
#include <sys/time.h>

int main()
{
  int    k;
  double xsum = 0.0;

  double t1, t2;
  double delta;
  struct timeval bg, dn;

  gettimeofday(&bg, NULL);
  for (k = 0; k < 10000000; k++)
    xsum += (double)k;
  gettimeofday(&dn, NULL);

  /* convert time to seconds, then calculate the difference */
```

```
  t1    = (double)bg.tv_sec + (double)bg.tv_usec/1000000.0;
  t2    = (double)dn.tv_sec + (double)dn.tv_usec/1000000.0;
  delta  = t2 - t1;

  printf("delta  = %10.6lf\n", delta);

  return 0;
}
```

Source Listing for GMGH

The source code for GMGH is divided into two source files, gmgh.c and gmgh.h. Both are listed below. Call out boxes highlight key elements of this code. This is designed to help the reader grasp the overall concept of this code. Compare this with the example at the end of Section 8.5.1, “The GPFS Multiple Access Range hints API” on page 161.

gmgh.c

```
/*********************************************************************/
/* Title    - Generic Middle-layer GPFS Hint package                 */
/* Module   - gmgh.c                                                 */
/* Envir.   - GPFS 1.4, VAC 5.0.1, AIX 4.3.3                         */
/* Last Mod -                                                        */
/*                                                                   */
/* ABSTRACT:                                                         */
/*   This is a generic middle layer code encapsulating the GPFS      */
/*   multiple access hint facility.  The native GPFS interface is    */
/*   overly tedius for use in high level application codes.  This    */
/*   code creates a simpler to use interface for high level programs.*/
/*                                                                   */
/*   The use of the multiple access hint facility in GPFS 1.3 and    */
/*   higher can significantly improve I/O performance of programs    */
/*   whose I/O access patterns are random.  The to use this code the */
/*   high level programmer must post up to MAXHINT hints in advance  */
/*   followed later by the data transfers (i.e., read or write)      */
/*   operations.  The comment headers for each function describe the */
/*   use of this code in detail.  See also ibm_phw.c and ibm_phr.c   */
/*   for examples of how to use this code.                           */
/*                                                                   */
/* FUNCTIONS AND PARAMETERS:                                         */
/*   gmgh_init_hint(p, fd, maxbsz, maxhint)                          */
/*   gmgh_post_hint(p, soff, nbytes, nth, isWrite)                   */
/*   gmgh_declare_1st_hint(p)                                        */
```

```
/*   gmgh_xfer(p, buf, nth)                                          */
/*   gmgh_gen_blk(p, nth, isWrite)                                   */
/*   gmgh_issue_hint(p, nth)                                         */
/*   gmgh_cancel_hint(fd)                                            */
/*                                                                   */
/*   buf     = caller provided buffer to deliver/receive data        */
/*             type - char*;  input, output                          */
/*   fd      = file descriptor                                       */
/*             type - int;  input, output                            */
/*   isWrite = boolean:  1 -> write, 0 -> read                       */
/*             type - int;  input                                    */
/*   maxbsz  = max buffer size;  maxbsz <= nbytes                    */
/*             type - int; input                                     */
/*   maxhint = max entries in hint vector                            */
/*             type - int; input                                     */
/*   nbytes  = number of bytes in record written/read                */
/*             type - int;  input                                    */
/*   nth     = ordinal value refering to the nth element             */
/*             type - int;  input                                    */
/*   p       = gmgh structure;  see function headers for details     */
/*             type - gmgh*;  input, output                          */
/*   soff    = 64 bit seek offset for record to be read/written      */
/*             type - off_t;  input                                  */
/*                                                                   */
/* COMPILATION FLAG PARAMETER:                                       */
/*   DEBUG - print an illustrative record to stdout of the accepted  */
/*           and released hints.                                     */
/*                                                                   */
/* WARNINGS AND CAVEATES:                                            */
/*   Error and warning messages printed directly to stdout.          */
/*********************************************************************/
```

Don't forget the header file.

```
#include "gmgh.h"

/***************************** FUNCTION ******************************/
/* Purpose:                                                          */
/*   "Mallocate" memory for the gmgh structures and initialize       */
/*   several fields in the structure.  maxbsz should be set large    */
/*   enough to handle the largest record that will be accessed.      */
/*   maxhint specifies the maximum number of records that will be    */
/*   posted as hints prior to accessing them.  There is a large      */
/*   arbitrary upper bound placed on these parameters to prevent     */
/*   careless mistakes.  Note, however, that the GPFS hint facility  */
/*   supporting this feature has been optimized for small record,    */
/*   randomly distributed accesses.  So when selecting maxbsz larger */
/*   than a GPFS block size caution should be used as performance    */
/*   may decrease in some cases.  See gmgh_post_hint() comment       */
/*   header for further details.                                     */
/* Parameters:                                                       */
```

```
/*   (I,O) gmgh *p     - gmgh structure.  The caller provides an     */
/*                       uninstantiated pointer;  this function      */
/*                       instantiates it by mallocating memory for   */
/*                       it and several substructures as well as     */
/*                       initializing some fields in the structure.  */
/*   (O)     p->blklst - mallocate block list                        */
/*   (O)     p->fd     - initialize file descriptor                  */
/*   (O)     p->gbsz   - initialize GPFS block size                  */
/*   (O)     p->hint   - mallocate hint vector                       */
/*   (O)     p->nbleh  - initialize max number of block list entries */
/*                       per hint                                    */
/*   (O)     p->nhve   - initialize max number of hint vector entries*/
/*   (I)   fd          - file descriptor                             */
/*   (I)   maxbsz      - max buffer size                             */
/*   (I)   maxhint     - max entries in hint vector                  */
/* Return:                                                           */
/*   Upon success, 0 will be returned, otherwise -1 is returned.     */
/*********************************************************************/
int gmgh_init_hint
(
   gmgh *p,                        /* gmgh structure */
   int  fd,                        /* file descriptor */
   int  maxbsz,                    /* max buffer size */
   int  maxhint                    /* max entries in hint vector */
)
{
   struct stat64 sbuf;
   gmgh_hint_t   *hvec;
   blklst_t      *blst;

   /*-----------------------------------------------*/
   /* Set a few parameters needed for calculations. */
   /*-----------------------------------------------*/

   p->fd = fd;

   if (fstat(p->fd, &sbuf) != 0)
   {
      printf("*** ERROR *** fstat error:  errno = %d\n", errno);
      return -1;
   }
   p->gbsz    = sbuf.st_blksize;

   p->nbleh   = maxbsz / p->gbsz;       /* validated below */
   if (maxbsz % p->gbsz > 0) p->nbleh++;

   p->nhve    = maxhint;                /* validated below */

   /*-----------------------------------------------*/
```

```
   /* Be sure we are starting with a clean slate. */
   /*---------------------------------------------*/

   if (gmgh_cancel_hint(p->fd) < 0)     /* error message already printed */
      return -1;

   /*---------------------------------------------------------------------*/
   /* The following limits are artificial.  Practically speaking, they  */
   /* are quite large and merely are trying to prevent wasting space if */
   /* not carefully defined.                                              */
   /*---------------------------------------------------------------------*/

   if (p->nbleh > MAXBLK)
   {
      printf("*** ERROR *** Hint buffersize too big.  See gmgh.h.\n");
      return -1;
   }

   if (p->nhve > MAXHINT)
   {
      printf("*** ERROR *** Too many hints requested.  See gmgh.h.\n");
      return -1;
   }

   /*---------------------------------------------*/
   /* "Mallocate" memory for the hint structures. */
   /*---------------------------------------------*/

   if (!(hvec = (gmgh_hint_t*)malloc(p->nhve * sizeof(gmgh_hint_t))))
   {
      printf("*** ERROR *** malloc error:  errno = %d\n", errno);
      return -1;
   }
   p->hint = hvec;

   if (!(blst = (blklst_t*)malloc(p->nhve * p->nbleh * sizeof(blklst_t))))
   {
      printf("*** ERROR *** malloc error:  errno = %d\n", errno);
      return -1;
   }
   p->blklst = blst;

   return 0;
}

/***************************** FUNCTION ******************************/
/* Purpose:                                                          */
/*   This function is called once for each record to be posted in a  */
/*   set of hints.  It tells gmgh which records are to be accessed   */
```

```
/*   in the future so that hints can be issued to GPFS (which       */
/*   results in prefetching records to cache). The maximum number of */
/*   hints that can be posted in a set must not exceed what was     */
/*   specified by gmgh_init_hint().  Each record posted here will   */
/*   later be accessed by gmgh_xfer() which will read or write the  */
/*   record as declared by isWrite.  However, one must not call     */
/*   gmgh_xfer() untill all the records for the hint set are posted. */
/*   After the last record is posted, gmgh_declare_1st_hint() must  */
/*   be called once and then gmgh_xfer() is called for each posted  */
/*   record.  The programmer should note that "under the covers"    */
/*   each record will be divided into blocks equal to the GPFS block */
/*   size if the record exceeds that size.  If the record size      */
/*   exceeds the max buffer size specified by gmgh_init_hint(), the */
/*   extra blocks will be ignored.  While there is no limit on the  */
/*   size of the record or number of hints that can be posted       */
/*   (beyond exceeding what was specified via gmgh_init_hint()),    */
/*   there is a limit to the number of GPFS blocks which will be    */
/*   accepted in a hint, but it is dependent upon the logical       */
/*   configuration of the disks.  gmgh will attempt to issue as many */
/*   posted hints prior to access as possible, but if the record    */
/*   size is quite large (e.g., multiple MB) not all of the GPFS    */
/*   blocks will be prefetched under the hint mechanism.  Again,    */
/*   this is handled automatcally by gmgh. In general, the hint     */
/*   facility is optimized to work with small record, randomly      */
/*   distributed accesses.                                          */
/* Parameters:                                                      */
/*   (I) gmgh *p - gmgh structure.                                  */
/*       (O) p->hint    - add nth hint to hint vector               */
/*       (I) p->nbleh   - max number of block list entries per hint */
/*       (I) p->nhve    - max number of hint vector entries         */
/*       (O) p->UBnblks - upper bound on number of used blocks      */
/*   (I) off_t soff  - seek offset                                  */
/*   (I) int nbytes  - record length in bytes                       */
/*   (I) int nth     - nth record in hint vector                    */
/*   (I) int isWrite - boolean; 0 = read, write = 1                 */
/* Return:                                                          */
/*   Upon success, 0 will be returned, otherwise -1 is returned.    */
/******************************************************************/
int gmgh_post_hint
(
   gmgh     *p,          /* gmgh structure */
   off_t    soff,        /* seek offset */
   int      nbytes,      /* record length in bytes */
   int      nth,         /* 0 <= nth < p->nhve */
   int      isWrite      /* read or write? */
)
{
   int status = 0;
```

```
   if (0 <= nth && nth < p->nhve)
   {
      if (nth == 0) p->UBnblks = 0;

      p->hint[nth].soff           = soff;
      p->hint[nth].len            = nbytes;
      p->hint[nth].lstblkreleased = 0;
      p->hint[nth].isWrite        = isWrite;

      p->UBnblks += p->nbleh;      /* upper bound on number of used blocks */

      if (gmgh_gen_blk(p, nth, isWrite) <= 0)    /* error has been printed */
         status = -1;
   }
   else
   {
      printf("*** ERROR *** invalid hint index;  nth = %d\n", nth);
      status = -1;
   }

   return status;
}

/***************************** FUNCTION ******************************/
/* Purpose:                                                          */
/*   Calling this function forces GPFS to prefetch as many of the    */
/*   records posted by gmgh_post_hint() as possible as well as       */
/*   resetting a few necessary GPFS structures and gmgh parameters.  */
/*   Records that can not be prefetched now will be prefetched       */
/*   automatically (if possible) when gmgh_xfer() is called. This is */
/*   done automatically.                                             */
/*                                                                   */
/*   Call this function after the last call to gmgh_post_hint() for  */
/*   the hint set and prior to the first call to gmgh_xfer() for the */
/*   hint set.  Failure to call this function in this manner will    */
/*   result in poor performance, without warning messages.           */
/* Parameters:                                                       */
/*   (I) gmgh *p - gmgh structure.                                   */
/*   (I)     p->fd     - initialize file descriptor                  */
/* Return:                                                           */
/*   Upon success, 0 will be returned, otherwise -1 is returned.     */
/*********************************************************************/
int gmgh_declare_1st_hint
(
   gmgh *p                        /* gmgh structure */
)
{
   int k;
```

Put records in the hint vector

```
   gmgh_cancel_hint(p->fd);       /* done once for each block of hints */

   /* Is there a way to enforce calling this in the correct order? */
   /* Probably yes.  Perhaps put some status variables in gmgh and */
   /* test/reset them PRN... but can this be made threadsafe?      */

   p->nxtblktoissue = 0;          /* do before first call to gmgh_issue_hint */

   k = gmgh_issue_hint(p, -1);    /* prefetch 1st records for future access */

   return k;
}

/***************************** FUNCTION ******************************/
/* Purpose:                                                          */
/*   Will either read or write the nth record in a hint set as       */
/*   specified by gmgh_post_hint() and will prefetch as many records */
/*   in advance of future "xfers" as possible.  This function should */
/*   be called once for each record in a hint set and each record    */
/*   should be "xfered" in the same order it was posted.             */
/* Parameters:                                                       */
/*   (I)   gmgh *p - gmgh structure.                                 */
/*         (I) p->fd   - initialize file descriptor                  */
/*         (I) p->hint - hint vector                                 */
/*   (I,O) buf     - caller provided data buffer                     */
/*   (I)   nth     - nth record in hint vector                       */
/* Return:                                                           */
/*   Upon success, 0 will be returned, otherwise -1 is returned.     */
/*********************************************************************/
int gmgh_xfer
(
   gmgh *p,              /* gmgh structure */
   char *buf,            /* externally provided data buffer */
   int  nth              /* nth record in hint vector;  0 <= nth < p->nhve */
)
{
   int nb;
   int status = 0;

   if (0 <= nth && nth < p->nhve)
   {
      if (lseek(p->fd, p->hint[nth].soff, SEEK_SET) < 0)
      {
         printf("*** ERROR *** seek error: soff = %lld, errno = %d\n",
                p->hint[nth].soff, errno);
         return -1;
      }

      if (p->hint[nth].isWrite)
```

Here is where the record is actually accessed

```
         nb = write(p->fd, buf, p->hint[nth].len);
      else
         nb = read(p->fd, buf, p->hint[nth].len);

      if (nb != p->hint[nth].len)
      {
         printf("*** ERROR *** disk I/O error: nth = %d, nb = %d, errno= %d\n",
                nth, nb, errno);
         return -1;
      }
   }
   else
   {
      printf("*** ERROR *** invalid hint index;  nth = %d\n", nth);
      return -1;
   }
```

Now issue the next hints

```
   status = gmgh_issue_hint(p, nth);    /* prefetch records for future access*/

   return status;
}

/***************************** FUNCTION ******************************/
/* Purpose:                                                          */
/*   For each record to be read or written, calculate the number of  */
/*   GPFS blocks touched and prepare a "block list" description for  */
/*   each block required for the GPFS_MULTIPLE_ACCESS_RANGE hint.    */
/*   Note that each entry in the gmgh hint vector corresponds to one */
/*   record in a read or write access and that this function will    */
/*   process only entry at a time.                                   */
/*                                                                   */
/*   This is a "private" function and should only be called in gmgh. */
/* Parameters:                                                       */
/*   (I) gmgh *p     - gmgh structure.                               */
/*       (O)   p->blklst - add block(s) to block list                */
/*       (I)   p->fd     - file descriptor                           */
/*       (I)   p->gbsz   - initialize GPFS block size                */
/*       (I,O) p->hint   - hint vector                               */
/*       (I)   p->nbleh  - max number of block list entries per hint */
/*   (I) int nth     - nth record in hint vector                     */
/*   (I) int isWrite - boolean; 0 = read, write = 1                  */
/* Return:                                                           */
/*   Upon success, the number of blocks touched is returned and      */
/*   should be greater than 0.  Returning a value <= 0 indicates are */
/*   error ocurred.                                                  */
/*********************************************************************/
int gmgh_gen_blk
(
   gmgh *p,                        /* gmgh structure */
```

```
   int  nth,                        /* nth entry;  0 <= nth < p->nhve */
   int  isWrite                     /* 1 - write access, 0 - read access */
)
{
   int      i;                      /* loop variables */
   int      nbt;                    /* number of blocks touched */
   int      blkidx;                 /* block index */
   int      disp;                   /* displacement */
   off_t    bn;                     /* block number */
   int      fblen, lblen;           /* first/last block's length */

   /*------------------------------*/
   /* How many blocks are touched? */
   /*------------------------------*/

   /* if the record touches more than p->nbleh blocks, ignore extra blocks */

   nbt  = (p->hint[nth].len + p->hint[nth].soff % p->gbsz - 1)/p->gbsz + 1;
   nbt  = (nbt <= p->nbleh) ? nbt : p->nbleh;

   p->hint[nth].nblkstouched = nbt;

   /*--------------------------------------*/
   /* Calculate parameters for first block. */
   /*--------------------------------------*/

   /* displacement to record in first GPFS block */

   disp = (int)(p->hint[nth].soff % (off_t)p->gbsz);

   /* first GPFS block touched by record */

   bn   = p->hint[nth].soff/(off_t)p->gbsz;

   /* length of first and last block in record */

   if (nbt == 1)
      fblen = p->hint[nth].len;
   else
   {
      fblen = p->gbsz - disp;
      lblen = (disp + p->hint[nth].len) - ((nbt - 1) * p->gbsz);
   }

   /*--------------------------------------*/
   /* Set parameters for each block touched. */
   /*--------------------------------------*/

   /*** get the first block's index ***/
```

Calculate how many blocks are spanned by the record

```
   blkidx               = nth * p->nbleh;
   p->hint[nth].fblkidx = blkidx;

   /*** do first block outside loop since its a little different ***/

   p->blklst[blkidx].blkoff  = disp;
   p->blklst[blkidx].blknum  = bn;
   p->blklst[blkidx].blklen  = fblen;
   p->blklst[blkidx].isWrite = isWrite;

   /*** do rest of the blocks if they exist ***/

   for (i = 1; i < nbt; i++)
   {
      p->blklst[blkidx + i].blkoff  = 0;
      p->blklst[blkidx + i].blknum  = bn + i;
      p->blklst[blkidx + i].isWrite = isWrite;

      if (i < nbt - 1)
         p->blklst[blkidx + i].blklen = p->gbsz;
      else
         p->blklst[blkidx + i].blklen = lblen;
   }

   /*** pad the unused blocks for variable length records ***/

   for (i = nbt; i < p->nbleh; i++)
   {
      p->blklst[blkidx + i].blkoff  = -1;
      p->blklst[blkidx + i].blknum  = 0;
      p->blklst[blkidx + i].blklen  = 0;
      p->blklst[blkidx + i].isWrite = 0;
   }

   return nbt;                      /* nbt > 0 => everything's OK */
}

/***************************** FUNCTION ******************************/
/* Purpose:                                                          */
/*   This function will issue the GPFS_MULTIPLE_ACCESS_RANGE hints   */
/*   via gpfs_fcntl().  It will take the hints as expanded in        */
/*   p->blklst and cancel the hints for the issued GPFS blocks       */
/*   corresponding to the nth entry in p->hint (i.e., the nth record */
/*   in the hint set which is presumably the latest record accessed).*/
/*   In the same call, it will also issue as many new hints from     */
/*   p->blklst as possible (in sets of GPFS_MAX_RANGE_COUNT).  Once  */
/*   no more new hints are accepted, it will stop issuing new hints  */
/*   and will re-issue any new hints that were not accepted in the   */
```

Find the hint blocks corresponding to the record and put them in the block list

```
/*   last set.  This function should be called once for each record  */
/*   (as specified by the parameter nth), just after its been        */
/*   accessed, and once before the first record in the hint set is   */
/*   accessed (in this case, nth = -1).                              */
/*                                                                   */
/*   This is a "private" function and should only be called in gmgh. */
/*                                                                   */
/* WARNING:  This code is subtle and prone to "off by one" bugs.     */
/*           use caution when modifying this.                        */
/*                                                                   */
/* Parameters:                                                       */
/*   (I) gmgh *p     - gmgh structure.                               */
/*       (I)   p->blklst - block list                                */
/*       (I)   p->fd     - file descriptor                           */
/*       (I)   p->hint   - hint vector                               */
/*       (I)   p->nbleh  - max number of block list entries per hint */
/*       (I)   p->nhve   - max number of hint vector entries         */
/*       (I,O) p->nxtblktoissue - next block to issue                */
/*       (I)   p->UBnblks - upper bound on number of used blocks     */
/*   (I) int nth     - nth record in hint vector                     */
/* Return:                                                           */
/*   Upon success, 0 will be returned, otherwise -1 is returned.     */
/*********************************************************************/
int gmgh_issue_hint
(
   gmgh *p,                   /* gmgh structure */
   int  nth                   /* hint index to release; -1 <= nth < p->nhve*/
)
{
   int    k, kk;                   /* loop variables */
   int    rem;                     /* remember a value */
   int    accDone, relDone;        /* while loop conditions */
   int    nhntacc;                 /* number of hints accepted */
   int    rbx;                     /* block list index for released blocks */
   int    ibx;                     /* block list index for issued blocks */

   struct
   {
      gpfsFcntlHeader_t hdr;
      gpfsMultipleAccessRange_t marh;
   } ghint;

   /*-----------------------*/
   /* Initialization stuff. */
   /*-----------------------*/

   ghint.marh.structLen  = sizeof(ghint.marh);
   ghint.marh.structType = GPFS_MULTIPLE_ACCESS_RANGE;
```

```
rbx = nth * p->nbleh;           /* first block in block list to release */

if (nth < -1 || nth >= p->nhve)
{
   printf("*** ERROR *** invalid nth parameter:  nth = %d\n", nth);
   return -1;
}

if (nth == -1) relDone = TRUE;  /* no hints to release */
else           relDone = FALSE;

accDone = FALSE;

/*--------------------------------------------------------------------*/
/* Release all blocks associated with the last accessed record, and */
/* continue issuing hints until no more are accepted or exist.      */
/*--------------------------------------------------------------------*/

while(!accDone || !relDone)
{
   /*** prepare data structure to release hints for accessed blocks ***/

   for (k = 0; k < GPFS_MAX_RANGE_COUNT; k++)
   {
      if (p->hint[nth].lstblkreleased >= p->hint[nth].nblkstouched)
      {
         relDone = TRUE;
         break;
      }

      ghint.marh.relRangeArray[k].blockNumber = p->blklst[rbx].blknum;
      ghint.marh.relRangeArray[k].start       = p->blklst[rbx].blkoff;
      ghint.marh.relRangeArray[k].length      = p->blklst[rbx].blklen;
      ghint.marh.relRangeArray[k].isWrite     = p->blklst[rbx].isWrite;

      rbx++;
      p->hint[nth].lstblkreleased++;
   }
   ghint.marh.relRangeCnt = k;

   /*** in case p->hint[nth].nblkstouched % GPFS_MAX_RANGE_COUNT == 0 ***/

   if (p->hint[nth].lstblkreleased >= p->hint[nth].nblkstouched)
      relDone = TRUE;

   /*** if we get behind, don't issue hints for released records ***/

   if (p->nxtblktoissue <= rbx)
      p->nxtblktoissue = p->nbleh * (nth + 1);
```

Prepare list of hint blocks to be released

```
/*** prepare data structure to issue hints for future accesses ***/

rem = p->nxtblktoissue;
ibx = p->nxtblktoissue;
k   = 0;

while (!accDone                &&
       k < GPFS_MAX_RANGE_COUNT &&
       ibx < p->UBnblks          )
{
   if (p->blklst[ibx].blkoff >= 0)     /* Is the next entry OK? */
   {
      ghint.marh.accRangeArray[k].blockNumber = p->blklst[ibx].blknum;
      ghint.marh.accRangeArray[k].start       = p->blklst[ibx].blkoff;
      ghint.marh.accRangeArray[k].length      = p->blklst[ibx].blklen;
      ghint.marh.accRangeArray[k].isWrite     = p->blklst[ibx].isWrite;

      ibx++;
      k++;
   }
   else                            /* If not, find next good one. */
      ibx++;
}
ghint.marh.accRangeCnt = k;
kk                     = k;       /* needed for DEBUG segment below */
p->nxtblktoissue       = ibx;

/*** actually release/issue hints ***/

ghint.hdr.totalLength   = sizeof(ghint);
ghint.hdr.fcntlVersion  = GPFS_FCNTL_CURRENT_VERSION;
ghint.hdr.fcntlReserved = 0;

if (gpfs_fcntl(p->fd, &ghint) != 0)
{
    printf("*** ERROR *** gpfs_fcntl error:  errno = %d\n", errno);
    return -1;
}
else
   nhntacc = ghint.marh.accRangeCnt;

#ifdef DEBUG
   for (k = 0; k < ghint.marh.relRangeCnt; k++)
      printf("R --> k = %d, blockNumber =%lld, start =%d, length =%d\n", k,
           ghint.marh.relRangeArray[k].blockNumber,
           ghint.marh.relRangeArray[k].start,
           ghint.marh.relRangeArray[k].length);
   printf("\n");
```

Prepare list of hint blocks to be issued

Actually issue and release hint blocks

Prints record of released and issued blocks which is helpful in learning how this code works

```
      for (k = 0; k < kk; k++)
         printf("A --> k = %d, blockNumber =%lld, start =%d, length =%d\n", k,
              ghint.marh.accRangeArray[k].blockNumber,
              ghint.marh.accRangeArray[k].start,
              ghint.marh.accRangeArray[k].length);
      printf("hints accepted = %d\n", nhntacc);
      printf(".........\n");
#endif

      /*** if all hints are not accepted, back track to last accepted hint ***/

      if (nhntacc < GPFS_MAX_RANGE_COUNT)
      {
         accDone = TRUE;           /* no more hints this time */

         ibx = rem - 1;
         k   = 0;
         while (k < nhntacc)
         {
            if (p->blklst[++ibx].blkoff < 0)    /* find next real data */
               continue;
            k++;
         }
         p->nxtblktoissue = ibx + 1;
      }
   }

   return 0;
}

/***************************** FUNCTION ******************************/
/* Purpose:                                                          */
/*    Remove any hints that have been issued against the open file   */
/*    p->fd.   This restores the hint status to what it was when the */
/*    file was first opened, but it does not alter the status of the */
/*    GPFS cache.                                                    */
/*                                                                   */
/*   This is a "private" function and should only be called in gmgh. */
/*                                                                   */
/* Parameters:                                                       */
/*   (I) gmgh *p     - gmgh structure.                               */
/*       (I) p->blklst - block list                                  */
/*       (I) p->fd     - file descriptor                             */
/*       (I) p->hint   - hint vector                                 */
/*       (I) p->nbleh  - max number of block list entries per hint   */
/*       (I) p->nhve   - max number of hint vector entries           */
/*       (O) p->nxtblktoissue - next block to issue                  */
/*       (I) p->UBnblks - upper bound on number of used blocks       */
/*   (I) int nth     - nth record in hint vector                     */
```

Figure out which hints were not accepted

```
/* Return:                                                            */
/*   Upon success, 0 will be returned, otherwise -1 is returned.      */
/*********************************************************************/
int gmgh_cancel_hint(int fd)       /* file descriptor */
{
   struct
   {
      gpfsFcntlHeader_t hdr;
      gpfsCancelHints_t cancel;
   } cancel;

   cancel.hdr.totalLength   = sizeof(cancel);
   cancel.hdr.fcntlVersion  = GPFS_FCNTL_CURRENT_VERSION;
   cancel.hdr.fcntlReserved = 0;
   cancel.cancel.structLen  = sizeof(gpfsCancelHints_t);
   cancel.cancel.structType = GPFS_CANCEL_HINTS;


   if (gpfs_fcntl(fd, &cancel) < 0)
   {
      printf("*** ERROR *** gpfs_fcntl error:  errno = %d\n", errno);
      return -1;
   }

   return 0;
}
```

gmgh.h

```
/*********************************************************************/
/* Title  - Generic Middle-layer GPFS Hint package                   */
/* Module - gmgh.h                                                   */
/*                                                                   */
/* See comments in gmgh.c for details.                               */
/*********************************************************************/

#ifndef GMGH_H
#define GMGH_H

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <memory.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <gpfs_fcntl.h>
```

```
#define FALSE 0
#define TRUE  1

#define GMGH_HINT_READ  0     /* Convenient parameters defined for user; */
#define GMGH_HINT_WRITE 1     /* to be used in gmgh_post_hint.           */

#define MAXBLK    256         /* max blklst size;  reclen <= MAXBLK * 256KB */
#define MAXHINT   128         /* max number of gmgh hints */

typedef struct                /* block descriptor -- nbleh blocks per hint */
{
   int       blkoff;          /* offset in bytes to start of record in block */
   off_t     blknum;          /* GPFS block number -- 1 block = 256KB */
   int       blklen;          /* lenght in bytes of record in block */
   int       isWrite;         /* write access = TRUE, otherwise FALSE */
} blklst_t;

typedef struct                /* one entry per record */
{
   off_t     soff;            /* 64 bit seek offset in bytes */
   int       len;             /* record length in bytes */
   int       nblkstouched;    /* number of blocks touched by record */
   int       lstblkreleased;  /* last block released from hint */
   int       isWrite;         /* write access = TRUE, otherwise FALSE */
   int       fblkidx;         /* this hint's 1st entry in block list */
} gmgh_hint_t;

typedef struct
{
   int         fd;            /* file descriptor */
   int         gbsz;          /* GPFS block size */
   int         nbleh;         /* max number of block list entries per hint */
   int         nhve;          /* max number of hint vector entries */
   int         UBnblks;       /* upper bound on number of used blocks */
   gmgh_hint_t *hint;         /* hint vector -- 1 entry per record */
   int         nxtblktoissue; /* next block to issue as hint in block list */
   blklst_t    *blklst;       /* block list -- each hint has 1 or more blocks
*/
} gmgh;

int gmgh_init_hint(gmgh*, int, int, int);
int gmgh_post_hint(gmgh*, off_t, int, int, int);
int gmgh_declare_1st_hint(gmgh*);
int gmgh_xfer(gmgh*, char*, int);
int gmgh_gen_blk(gmgh*, int, int);
int gmgh_issue_hint(gmgh*, int);
int gmgh_cancel_hint(int);

#endif
```

Here is the block list structure

Here is the hint vector structure

Here is the general gmgh structure which contains the hint vector and block list

H

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246035

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG24-6035.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
README.ibm_seq	This is an ASCII file containing instructions regarding the ibm_seq.tar file.
ibm_seq.tar	This is a tar file containing the source code, makefile and ksh scripts for the sequential benchmark programs discussed in the book.
README.ibm_par	This is an ASCII file containing instructions regarding the ibm_par.tar file.
ibm_par.tar	This is a tar file containing the source code, makefile, ksh scripts and hostfiles for the parallel benchmark codes corresponding to the sequential benchmark codes discussed in the book.
gpfs.utilities.tar	This tar file contains the scripts detailed in Appendix C, “A useful tool for distributed commands” on page 211 and Appendix D, “Useful scripts” on page 217.

System requirements for downloading the Web material

There are few specific requirements for downloading this web material. The specific recommendations are:

Hard disk space: 0.5 MB
Operating System: UNIX

How to use the Web material

Create a unique subdirectory for each tar file in an AIX RS 6000 system and transfer the desired tar file to the appropraite subdirectory. If you use ftp to do this, be sure to set the file transfer type to binary for the tar files. Next untar the files and make the executables. The README.<ext> file provides detailed instructions regarding the system requirements and usage of these programs.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications see , “How to get IBM Redbooks” on page 263.

- *Exploiting HACMP 4.4: Enhancing the Capabilities of Cluster Multi-Processing, SG24-5979*
- *GPFS: A Parallel File System,* SG24-5165
- *Monitoring and Managing IBM SA Disk Subsystems,* SG24-5251
- *A Practical Guide to Serial Storage Architecture for AIX,* SG24-4599
- *PSSP Version 3 Survival Guide, SG24-5344*
- *RSCT Group Services: Programming Cluster Applications,* SG24-5523
- *Sizing and Tuning GPFS,* SG24-5610
- *Understanding SSA Subsystems in Your Environment,* SG24-5750

Other resources

These publications are also relevant as further information sources:

- *7133 SSA Subsstem: Hardware Technical Information*, SA33-3261
- *7133 SSA Subsystem: Operator Guide,* GA33-3259
- *7133 Models 010 and 020 SSA Disk Subsystems: Installation Guide*, GA33-3260
- *7133 Models 500 and 600 SSA Disk Subsystems: Installation Guide*, GA33-3263
- *7133 SSA Disk Subsystem: Service Guide*, SY33-0185
- *GPFS for AIX: Guide and Reference,* SA22-7452
- *GPFS for AIX: Installation and Tuning Guide,* GA22-7453
- *GPFS for AIX: Problem Determination Guide,* GA22-7434
- *GPFS for AIX: Concepts, Planning, and Installation Guide*, GA22-7453

- *HACMP 4.3: Enhanced Scalability Installation and Administration Guide,* Vol.1, SC23-4284-02
- *HACMP 4.3: Enhanced Scalability Installation and Administration Guide,* Vol.2, SC23-4306-01
- *Micro Channel SSA RAID Adapters Technical Reference*, SA33-3270
- *PCI SSA RAID Adapters: Technical Reference*, SA33-3225
- *Planning SSA RAID Subsystems,* GA33-3271
- *POSIX Programmer's Guide,* Donald Levine, O'Reilly and Associates, April 1991, ISBN 0-937175-73-0
- *PSSP: Administration Guide*, SA22-7348
- *PSSP: Managing Shared Disks*, SA22-7349
- *RSCT: Group Services Programming Guide and Reference,* SA22-7355
- *SSA Adapters: Technical Reference,* S31H-8612
- *SSA Adapters: User's Guide and Maintenance Information*, SA33-3272

Referenced Web sites

These Web sites are also relevant as further information sources:

- `http://www.rge.com/pub/systems/aix/bull/`

  This is a collection of freeware and shareware prepackaged for installation on AIX based systems. It contains GNU and other offerings. Its maintained by Group Bull.

- `http://www.gnu.org/`

  This is the GNU web page. If a GNU offering is not in the Group Bull listing, it can be obtained here, though a little more work will be required to install it. In addition to its own offerings, the GNU web site has pointers to numerous other sites with free software.

- `http://home.flash.net/~marknu/less/download.html`

  In doing the concomitant lab work associated with writing this book, the authors downloaded a helpful tool called less from this web site. less is a pager program which uses the vi commands for scrolling and has other useful pager tool functions.

- `http://www.hursley.ibm.com/ssa/index.html`

  This site provides current information on IBM SSA adapters and disk drives including compatibility, microcode and device drivers.

- `http://techsupport.services.ibm.com/rs6000/support`

  A place to start when updating your RS/6000 host with fixes, drivers and tools.

- `http://www.rs6000.ibm.com/resource/technology/gpfs_perf.html`

  GPFS performance White Paper

- `http://www.cae.de.ibm.com/forum/ssa/ssa.forum.html`

  This site carries discussions on SSA and can provide additional links to other useful sites.

How to get IBM Redbooks

Search for additional Redbooks or redpieces, view, download, or order hardcopy from the Redbooks Web Site

ibm.com/redbooks

Also 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.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web Site for information about all the CD-ROMs offered, updates and formats.

Special notices

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere.,The Power To Manage., Anything. Anywhere.,TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

block. The largest contiguous segment of data in a GPFS file. It is set when the file system is created using the **mmcrfs** command and can not be reset using the **mmchfs** command. Data is commonly accessed in units of blocks, but under some circumstances only subblocks are accessed.

cluster. A loosely-coupled set of nodes organized into a network for the purpose of sharing resources and communicating with each other. See GPFS cluster.

concurrent access. Simultaneous access to a shared volume group or a raw disk by two or more nodes. In this configuration, all the nodes defined for concurrent access to a shared volume group are owners of the shared resources associated with the volume group or raw disk.

If one of the nodes in a concurrent access environment fails, it releases the shared volume group or disk, along with its resources. Access to the shared volume group or disk is, however, continuously available, as long as another node is up. Applications can switch to another server immediately.

configuration manager. A GPFS global management function assigned to the longest running node in the file system group as monitored by group services. It selects file system manager nodes and guarantees quorum exists.

datashipping. A GPFS optimization where one node accesses records from another node's GPFS cache. Using this prevents the node with the record from having to flush its copy to cache and the requesting node to fetch it from disk. The efficacy of this optimization is limited by the fact that the data must be transferred between nodes; while this is faster than flushing to and fetching from disk, it is not as fast as a node accessing its own cache. This optimization is useful for applications which perform fine grained writes across multiple nodes or which append data to the end of a file from multiple nodes. Datashiping is a GPFS directive.

directive. See GPFS directive.

disk descriptor. A disk descriptor defines how a disk is to be used within a GPFS file system. Each descriptor must be in the form:

```
DiskName:::DiskUsage:FailureGroup
```

Where `DiskName` is the name of the disk. In a vsd environment this must be the virtual shared disk name. In either a GPFS cluster or non-vsd environment, this must be the name of the concurrent logical volume. `DiskUsage` tells GPFS whether data, metadata, or both are to be stored on the disk. The `FailureGroup` designation indicates to GPFS where not to place replicas of data and metadata. All disks with a common point of failure should belong to the same failure group. Since GPFS does not place replicated information on disks in the same failure group, the availability of information is ensured even in the event of disk failure.

failure group. A set of disks that share a common adaptor or access path and which

could become inaccessible by one hardware failure.

File System Manager. There is one File System Manager per file system. It assumes responsibility for file system configuration, disk space allocation management, token management, quota management and security services.

flush. Transfer file information in main memory to disk. The fsync() call does not return control to application program until this transfer is complete. It does not necessarily affect the contents of a file cache.

GPFS cluster. A subset of clustered nodes declared as being available to GPFS. See cluster and nodeset.

GPFS Access Range hint. A mechanism whereby an application program discloses a range of GPFS file blocks to be accessed in advance of their actual access. It assumes that all application program accesses are in the specified range.

GPFS directive. An action specified by the application program that GPFS must execute or else return an error. It is stronger than a GPFS hint.

GPFS hint. A disclosure of future file accesses to GPFS allowing prefetching and write-behind access when the pattern can not be automatically recognized. A hint is only a suggestion and is not guaranteed to be acted upon (and is weaker than a GPFS directive).

GPFS Multiple Access Range hint. A flexible mechanism whereby an application program discloses a range of GPFS file blocks to be accessed in advance of their actual access. The API allows this range to be released without altering the GPFS cache so that a new range can be specified.

GPFS system data (GSD). A repository of GPFS configuration data. It is stored in the SDR on an SP system and on the primary and secondary nodes of a cluster as defined by the `mmcrcluster` command.

GSD. See GPFS system data

hint. See GPFS hint.

i-node. Metadata stored on disk describing a file to AIX. It contains the file size, addresses of the physical data blocks or indirect blocks which in turn point to the physical data blocks comprising the file, etc.

Journaled File System (JFS). A sequential file system commonly used as the local file system in AIX RS6000 nodes.

Logical Volume Manager (LVM). Manages disk space at a logical level. It controls fixed-disk resources by mapping data between logical and physical storage, allowing data to be discontiguous, span multiple disks, replicated, and dynamically expanded.

MAR hint. See GPFS Multiple access range hint.

metadata. Information about a file. It is used by GPFS to describe the file and to locate its contents. It includes i-nodes, indirect blocks and other data.

metanode. The node responsible for maintaining the integrity of the metadata for an open file. It is generally the node on which the file has been open for the longest continuous period of time.

mirroring. The process of maintaining a duplicate copy of all disk data in order to preserve a file in the event of a disk failure. Mirroring is done at the logical volume

manager (LVM) level and is unrelated to GPFS replication.

mmfsd. The GPFS daemon.

multi-node quorum. A quorum algorithm used with three or more nodes. A quorum is defined as one plus half the number of nodes in a nodeset.

nodeset. A subset of nodes in a GPFS cluster which have access to the same file systems. If multiple nodesets are defined in a GPFS cluster, they must be disjoint.

quorum. The minimum number of nodes that must be running in a nodeset for the GPFS daemon to start.

quota. The maximum amount of disk space that can be used by a single user or group.

RAID. Redundant Array of Inexpensive Disks. A set of disks that act as a single physical volume using parity checking prevent data loss.

read-ahead. A read I/O operation optimization in GPFS where GPFS places records in its cache prior to their access. This allows a read() system call to return after fetching the record from memory without having to wait for it to be delivered from disk to memory first.

replication. A GPFS mechanism for creating and maintaining multiple file copies to ensure availability in the advent of a hardware failure. This is not related to LVM mirroring.

single-node quorum. A quorum algorithm used with two nodes. Only one node is needed for a quorum.

SSA. Serial Storage Architecture. An expanded storage adapter for multi-processor data sharing in UNIX-based computing, allowing disk connection in a high-speed loop.

stripe group. The set of disks comprising the storage assigned to a file system.

striping. The mechanism used by GPFS for writing files in parallel to multiple disks instead of to a single disk in a serial operation.

subblock. One thirty second of a block. It is the smallest unit of data accessible in a GPFS file operation, though data is more commonly transferred in whole blocks.

Token Management. A mechanism for controlling access to a shared file. A token is granted for a given byte range in a file and the process holding this token has exclusive use of that byte range as needed.

twin tailing. Connecting a disk to multiple nodes;

vnode. The data structure which contains information about a file system object in AIX.

Virtual Shared Disk. A component of PSSP commonly used on an SP system where nodes have access to remotely mounted raw logical volumes. It is commonly used as a server by GPFS clients. It is not available at this time in GPFS clusters.

VSD. See Virtual Shared Disk.

write-behind. A write I/O operation optimization in GPFS where a write() system call places a record in GPFS cache and the record is scheduled to be written to disk at some later, more optimum time. This allows the write() system call to return control to the application program without waiting for it to be physically placed on disk.

Abbreviations and acronyms

ANSI	American National Standards Institute
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
BOS	Base Operating System
BLOB	Binary Large Object
CLVM	Concurrent LVM
CSPOC	Cluster Single Point of Control
DARE	Dynamic Automatic Reconfiguration Event
DNS	Domain Name System
EOF	End of File
ESSL	Engineering & Scientific Subroutine Library
FCAL	Fibre Channel Arbitrated Loop
FTP	File Transfer Protocol
GMGH	Generic Middle layer GPFS Hint API
GPFS	IBM General Parallel File System for AIX
GSAPI	IBM Group Services API
HACMP	High Availability Cluster Multi-Processing
HACMP/ES	High Availability Cluster Multi-Processing/Enhanced Scalability
IBM	International Business Machines Corporation
ISO	International Organization for Standardization
IT	Information Technology
ITSO	International Technical Support Organization
I/O	Input/Output
JBOD	Just a Bunch Of Disks
JFS	Journaled File System
LAN	Local Area Network
LPP	Licensed Program Products
LRU	Least Recently Used
LV	Logical Volume
LVCB	Logical Volume Control Block
LVM	Logical Volume Manager
MAR	Multiple Access Range hint
NFS	Network File System
ODM	Object Data Manager
OEM	Original Equipment Manufacturer
OSF	Open Software Foundation, Inc.
PCI	Peripheral Component Interconnect
PID	Process ID
POSIX	Portable Operating System Interface
PSSP	IBM Parallel System Support Programs for AIX
PESSL	Parallel ESSL
PTF	Program Temporary Fix
PV	Physical Volume
PVID	Physical Volume Identification
RAID	Redundant Array of Independent Disks
RAS	Reliability Availability Serviceability

RSCT	IBM Reliable Scalable Cluster Technology
RVSD	IBM Recoverable Virtual Shared Disk
SAN	Storage Area Network
SCSI	Small Computer System Interface
SGID	Set Group ID
SMIT	System Management Interface Tool
SMP	Symmetric Multiprocessor
SSA	Serial Storage Architecture
TCP/IP	Transmission Control Protocol/Internet Protocol
UTP	Unshielded Twisted Pair
VFS	Virtual File System
VG	Volume Group
VGDA	Volume Group Descriptor Area
VGSA	Volume Group Status Area
VSD	IBM Virtual Shared Disk
VSDM	Virtual Shared Disk Manager
WebSM	Web-based System Manager

Index

C

D

E

F

G

H

M

N

O

P

Q

R

S

T

U

V

W

GPFS on AIX Clusters; High Performance File System Administration Simplified

GPFS on AIX Clusters:

High Performance File System Administration Simplified

Learn how to install and configure GPFS 1.4 step by step

Understand basic and advanced concepts in GPFS

Learn how to exploit GPFS in your applications

With the newest release of General Parallel File System for AIX (GPFS), release 1.4, the range of supported hardware platforms has been extended to include AIX RS/6000 workstations that are not part of an RS/6000 SP system. This is the first time that GPFS has been offered to non-RS/6000 SP users. Running GPFS outside of the RS/6000 SP does require that HACMP/ES is configured and that the RS/6000 systems within the HACMP cluster that will be part of the GPFS cluster be concurrently connected to an SSA disk subsystem.

This redbook focuses on the planning, installation and implementation of GPFS in a cluster environment. The tasks to be covered include the installation and configuration of HACMP to support the GPFS cluster, implementation of the GPFS software, and developing application programs that use GPFS. A troubleshooting chapter is added in case any problems arise.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6035-00 ISBN 0738422088