General Parallel File System

Advanced Administration Guide

Version 3 Release 2.1

SC23-5182-02

General Parallel File System

Advanced Administration Guide

Version 3 Release 2.1

SC23-5182-02

Note:
Before using this information and the product it supports, be sure to read the general information under “Notices” on page 131.

Third Edition (August 2008)

This edition applies to version 3, release 2, modification 1 of IBM General Parallel File System Multiplatform (product number 5724-N94), IBM General Parallel File System for POWER™ (product number 5765-G66), and to all subsequent releases and modifications until otherwise indicated in new editions. Technical changes or additions to the text and illustrations are indicated by a vertical line (|) to the left of the change.

IBM welcomes your comments. A form for your comments may be provided at the back of this publication, or you may send your comments to this address:

International Business Machines Corporation
Department 58HA, Mail Station P181
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States and Canada): 1+845+432-9405
FAX (Other Countries): Your International Access Code +1+845+432-9405

IBMLink™ (United States customers only): IBMUSM10(MHVRCFS)
Internet e-mail: mhvrcfs@us.ibm.com

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this publication
- Page number or topic related to your comment

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

Contents

Figures

Tables

About this information

This information explains how to use advanced function for the General Parallel File System™ (GPFS™) licensed product:

- Accessing GPFS file systems from other GPFS clusters
- Policy-based data management for GPFS
- Creating and maintaining snapshots of GPFS file systems
- Establishing disaster recovery for your GPFS cluster
- Monitoring GPFS I/O performance with the **mmpmon** command
- Miscellaneous advanced administration topics

This edition applies to GPFS version 3.2.1 for AIX®, Linux®, and Windows®.

To find out which version of GPFS is running on a particular AIX node, enter:

```
lslpp -l gpfs\*
```

To find out which version of GPFS is running on a particular Linux node, enter:

```
rpm -qa | grep gpfs
```

To find out which version of GPFS is running on a particular Windows node, use the graphical user interface (GUI) and follow these steps:

1. Click **Control Panel**–>**Add or Remove Programs**
2. Click **IBM General Parallel File System** and choose **Click here for support information.**

For updates to this information, see publib.boulder.ibm.com/infocenter/clresctr/topic/com.ibm.cluster.gpfs.doc/gpfsbooks.html.

For the latest support information, see the GPFS Frequently Asked Questions at publib.boulder.ibm.com/infocenter/clresctr/topic/com.ibm.cluster.gpfs.doc/gpfs_faqs/gpfsclustersfaq.html.

Who should read this information

This information is designed for system administrators and programmers seeking to understand and use the advanced features of GPFS. To use this information, you must be familiar with the GPFS licensed product and the AIX, Linux, or Windows operating system, or all of them, depending on which operating systems are in use at your installation.

Conventions used in this information

Table 1 describes the typographic conventions used in this information.

Table 1. Typographic conventions

Typographic convention	Usage
Bold	**Bold** words or characters represent system elements that you must use literally, such as commands, flags, path names, directories, file names, values, and selected menu options.
Bold Underlined	**Bold Underlined** keywords are defaults. These take effect if you fail to specify a different keyword.
Italic	• *Italic* words or characters represent variable values that you must supply. • *Italics* are also used for publication titles and for general emphasis in text.

Table 1. Typographic conventions (continued)

Typographic convention	Usage
`Constant width`	All of the following are displayed in `constant width` typeface: • Displayed information • Message text • Example text • Specified text typed by the user • Field names as displayed on the screen • Prompts from the system • References to example text
[ ]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
|	A vertical bar separates items in a list of choices. (In other words, it means ″or″)
< >	Angle brackets (less-than and greater-than) enclose the name of a key on the keyboard. For example, **<Enter>** refers to the key on your terminal or workstation that is labeled with the word Enter.
...	An ellipsis indicates that you can repeat the preceding item one or more times.
<Ctrl-*x*>	The notation **<Ctrl-*x*>** indicates a control character sequence. For example, **<Ctrl-*c*>** means that you hold down the control key while pressing **<c>**.
\	The continuation character is used in programming examples in this information for formatting purposes.

Prerequisite and related information

For updates to this information, see publib.boulder.ibm.com/infocenter/clresctr/topic/com.ibm.cluster.gpfs.doc/gpfsbooks.html.

For the latest support information, see the GPFS Frequently Asked Questions at publib.boulder.ibm.com/infocenter/clresctr/topic/com.ibm.cluster.gpfs.doc/gpfs_faqs/gpfsclustersfaq.html.

ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.

Using LookAt to look up message explanations

LookAt is an online facility that lets you look up explanations for most of the IBM® messages you encounter, as well as for some system abends and codes. You can use LookAt from the following locations to find IBM message explanations for Clusters software products:

- The Internet. You can access IBM message explanations directly from the LookAt Web site: http://www.ibm.com/systems/z/os/zos/bkserv/lookat/
- Your wireless handheld device. You can use the LookAt Mobile Edition with a handheld device that has wireless access and an Internet browser (for example, Internet Explorer for Pocket PCs, Blazer, or Eudora for Palm OS, or Opera for Linux handheld devices). Link to the LookAt Mobile Edition from the LookAt Web site.

How to send your comments

Your feedback is important in helping us to produce accurate, high-quality information. If you have any comments about this information or any other GPFS documentation:

- Send your comments by e-mail to: mhvrcfs@us.ibm.com.

  Include the publication title and order number, and, if applicable, the specific location of the information you have comments on (for example, a page number or a table number).
- Fill out one of the forms at the back of this information and return it by mail, by fax, or by giving it to an IBM representative.

To contact the IBM cluster development organization, send your comments by e-mail to: cluster@us.ibm.com.

Summary of changes

The following sections summarize changes to the GPFS licensed program and the GPFS library for version 3, release 2, modification 1. Within each information unit in the library, a vertical line to the left of text and illustrations indicates technical changes or additions made to the previous edition of the book.

Summary of changes
for GPFS Version 3, Release 2, Modification 1
as updated, August 2008

Changes to GPFS and to the GPFS library for version 3, release 2, modification 1 include:

- **New information**
  - GPFS for Windows Multiplatform, V3.2.1 supports the Windows Server 2003 R2 operating system running on 64-bit architectures (AMD x64 / EM64T). GPFS on Windows participates in a new or existing GPFS V3.2 cluster in conjunction with AIX and Linux (32- or 64-bit) operating systems.
  - Identity mapping between Windows and UNIX® user accounts is one of the key advancements delivered in GPFS for Windows Multiplatform. System administrators can explicitly match users and groups defined on UNIX with those defined on Windows. This allows users to maintain file ownership and access rights from either platform. System administrators are not required to define an identity map. GPFS automatically creates a mapping when one is not defined. For more information about identity mapping, see the *General Parallel File System: Concepts, Planning, and Installation Guide* and the *General Parallel File System: Advanced Administration Guide*.
  - IBM has enhanced many of the details within GPFS to support the unique semantic requirements posed by Windows. These include case insensitive names, NTFS-like file attributes, and Windows file locking. GPFS provides a bridge between a Windows and POSIX view of files, while not adversely affecting the long-standing capabilities provided on AIX and Linux operating systems.
  - GPFS for Windows Multiplatform provides the same core services to parallel and serial applications as are available on AIX and Linux operating systems. GPFS allows parallel applications simultaneous access to the same files, or different files, from any node that has the GPFS file system mounted while managing a high level of control over all file system operations. System administrators and users have a consistent command interface on AIX, Linux, and Windows operating systems.

    The following commands have been updated for Windows:
    - **mmchfs** to add the **-t** *DriveLetter* option
    - **mmcrfs** to add the **-t** *DriveLetter* option
    - **mmlsfs** to add the **-t** option to display the Windows drive letter
    - **mmmount** to add the *DefaultDriveLetter* and *DriveLetter* parameters
    - **mmumount** to add the *DefaultDriveLetter* and *DriveLetter* parameters

    With few exceptions, the commands supported on the Windows operating system are identical to the commands supported on other GPFS platforms. For a list of unsupported commands, see the *General Parallel File System: Concepts, Planning, and Installation Guide*.
  - GPFS for Windows Multiplatform, V3.2.1 does not support or has restricted support for some features. For a complete list of these limitations, see the *General Parallel File System: Concepts, Planning, and Installation Guide*.
- **Changed information:**

  Minor editorial updates marked by a vertical line to the left of the text.
- **Deleted information:**

  There has been no information deleted from the GPFS library for GPFS V3.2.1.

Chapter 1. Accessing GPFS file systems from other GPFS clusters

GPFS allows users shared access to files in either the cluster where the file system was created, or other GPFS clusters. File system access by the cluster where the file system was created is implicit.

The ability to access and mount GPFS file systems owned by other clusters in a network of sufficient bandwidth is accomplished using the **mmauth**, **mmremotecluster** and **mmremotefs** commands. Each site in the network is managed as a separate cluster, while allowing shared file system access.

The cluster owning the file system is responsible for administering the file system and granting access to other clusters on a per cluster basis. After access to a particular file system has been granted to nodes in another GPFS cluster, they may mount the file system and perform data operations as if the file system were locally owned.

Each node in the GPFS cluster requiring access to another cluster's file system must be able to open a TCP/IP connection to every node in the other cluster.

Nodes in two separate remote clusters mounting the same file system are no longer required to be able to open a TCP/IP connection to each other, which was the case in versions of GPFS prior to GPFS 3.1. For example, if a node in **clusterA** mounts a file system from **clusterB**, and a node in **clusterC** desires to mount the same file system, nodes in **clusterA** and **clusterC** do not have to communicate with each other.

Each node in the GPFS cluster requiring file system access must have either:

- A virtual connection to the file system data through an NSD server (refer to Figure 1).
- A physical connection to the disks containing file system data (refer to Figure 2 on page 2).

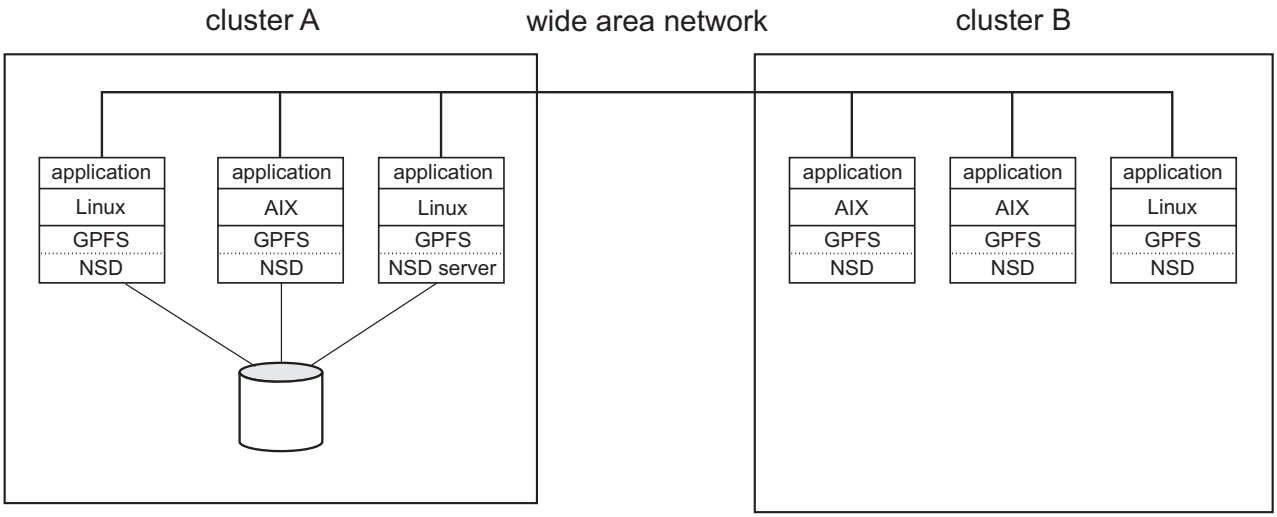


Figure 1. Remote mount of a file system using NSD server access

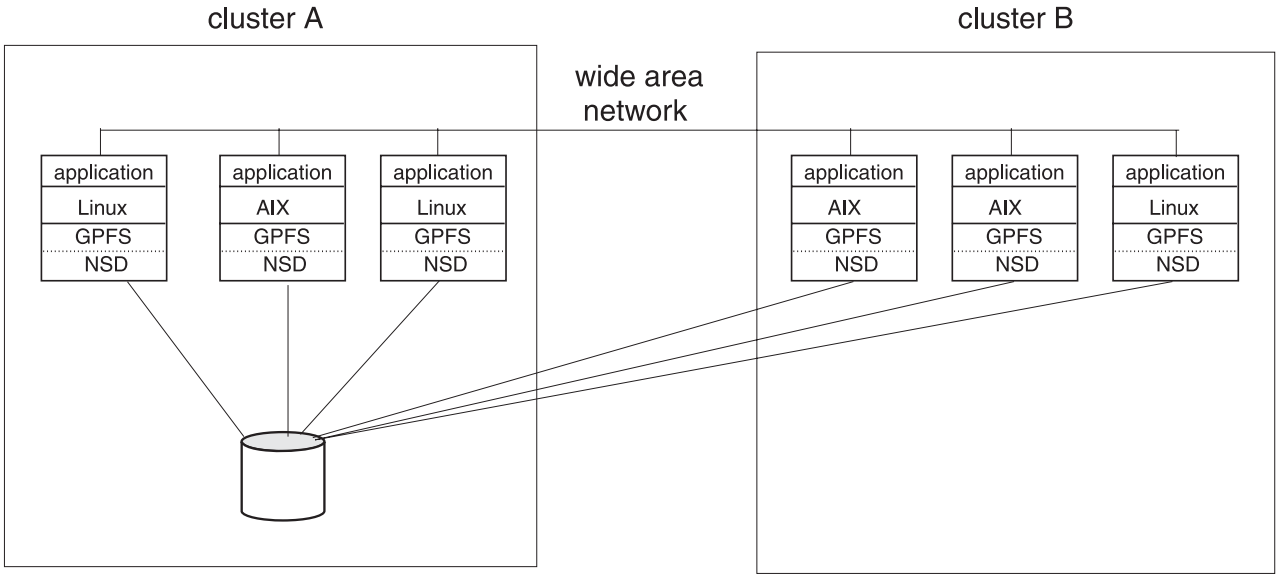


Figure 2. Remote mount of a file system using SAN-attached disks

Figure 3 on page 3 illustrates a multi-cluster configuration with multiple NSD servers. In this configuration:

- The two nodes in Cluster 1 are defined as the NSD servers (you can have up to eight NSD server nodes).
- All three clusters are connected with Gigabit Ethernet.
- Cluster 1 shares a Myrinet network with Cluster 2 and a High Performance Switch (HPS) network with Cluster 3.

In order to take advantage of the fast networks and to use the nodes in Cluster 1 as NSD servers for Cluster 2 and Cluster 3, you must configure a subnet for each of the supported clusters. For example issuing the command:

- **mmchconfig subnet=**″**<***MyrinetSubnet***> <***MyrinetSubnet***>/Cluster1**″ in Cluster 2 allows nodes N_2 through N_x to use N_1 as an NSD server with the Myrinet network providing the path to the data.
- **mmchconfig subnet=**″**<***FedSubnet***> <***FedSubnet***>/Cluster1**″ in Cluster 3 allows nodes N_{2+x} through N_{y+x} to use N_{1+x} as an NSD server with the HPS network providing the path to the data.

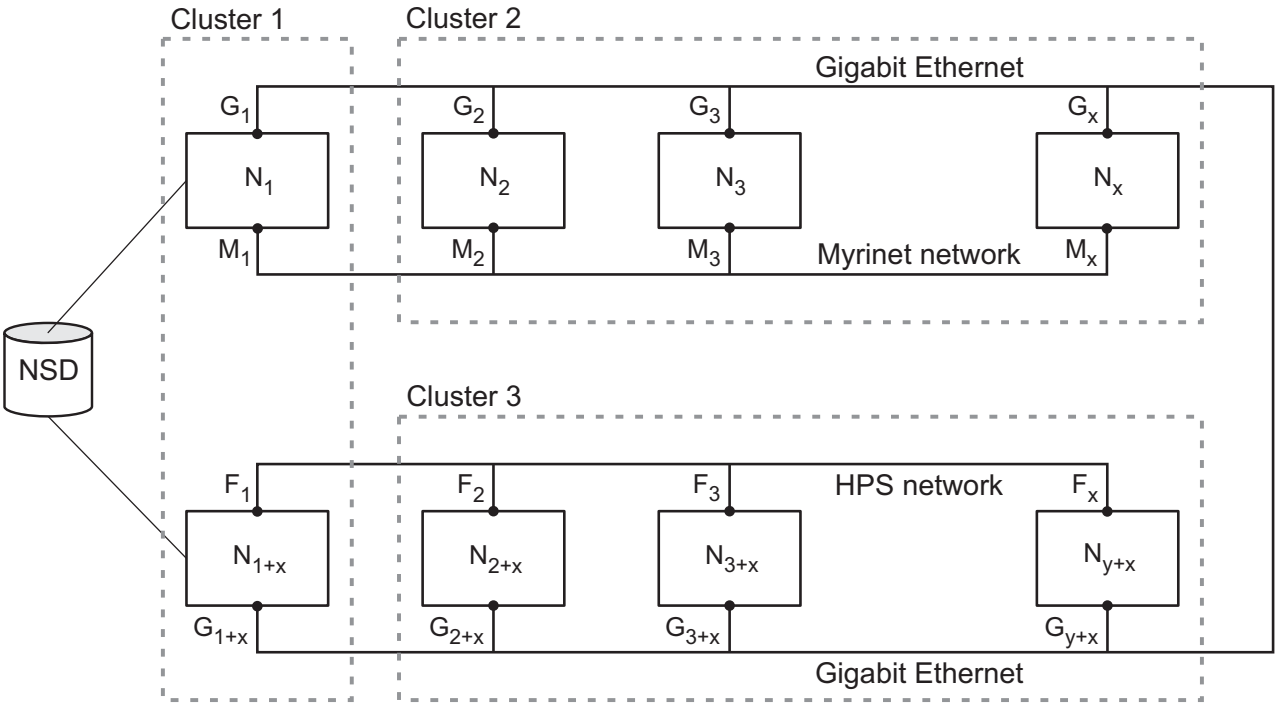


Figure 3. Multi-cluster configuration with multiple NSD servers

When you implement file access from other clusters, consider these topics:

- "User access to a GPFS file system owned by another GPFS cluster"
- "Mounting a file system owned and served by another GPFS cluster" on page 4
- "Managing remote access to GPFS file systems" on page 6
- "Using remote access with public and private IP addresses" on page 7
- "Using multiple security levels" on page 8
- "Changing security keys" on page 9
- "Additional information about GPFS file systems accessed by nodes that belong to other GPFS clusters" on page 10

User access to a GPFS file system owned by another GPFS cluster

In a cluster environment that has a single user identity name space, all nodes have user accounts set up in a uniform manner. This is usually accomplished by having equivalent **/etc/passwd** and **/etc/group** files on all nodes in the cluster.

For consistency of ownership and access control, a uniform user identity name space is preferred. For example, if user Jane Doe has an account on nodeA with the user name **janedoe** and user ID **1001** and group ID **500**, on all other nodes in the same cluster Jane Doe will have an account with the same user and group IDs. GPFS relies on this behavior to perform file ownership and access control tasks.

If a GPFS file system is being accessed from a node belonging to another GPFS cluster, the assumption about the uniform user account infrastructure may no longer be valid. Since different clusters may be administered by different organizations, it is possible for each of the clusters to have a unique set of user accounts. This presents the problem of how to permit users to access files in a file system owned and served by another GPFS cluster. In order to have such access, the user must be somehow known to the other cluster. This is usually accomplished by creating a user account in the other cluster, and giving this account the same set of user and group IDs that the account has in the cluster where the file system was created.

To continue with the example above, Jane Doe would need an account with user ID **1001** and group ID **500** created in every other GPFS cluster from which remote GPFS file system access is desired. This approach is commonly used for access control in other network file systems, (for example, NFS or AFS®), but might pose problems in some situations.

For example, a problem arises if Jane Doe already has an account in some other cluster, but the user ID associated with this account is not **1001**, and another user in the other cluster has user ID **1001**. It would require a considerable effort on the part of system administrator to ensure that Jane Doe's account has the same set of IDs on all clusters. It is more desirable to be able to use the existing accounts without having to make changes. GPFS helps to solve this problem by optionally performing user and group ID remapping internally, using user-supplied helper applications. A detailed description of the GPFS user ID remapping convention is contained in *UID Mapping for GPFS in a Multi-Cluster Environment* at www.ibm.com/servers/eserver/clusters/library/wp_aix_lit.html.

Access from a remote cluster by a root user presents a special case. It is often desirable to disallow root access from a remote cluster while allowing regular user access. Such a restriction is commonly known as root squash. A root squash option is available when making a file system available for mounting by other clusters using the **mmauth** command. This option is similar to the NFS root squash option. When enabled, it causes GPFS to squash superuser authority on accesses to the affected file system on nodes in remote clusters.

This is accomplished by remapping the credentials: user id (UID) and group id (GID) of the root user, to a UID and GID specified by the system administrator on the home cluster, for example, the UID and GID of the user nobody. In effect, root squashing makes the root user on remote nodes access the file system as a non-privileged user.

Although enabling root squash is similar to setting up UID remapping, there are two important differences:

1. While enabling UID remapping on remote nodes is an option available to the remote system administrator, root squashing need only be enabled on the local cluster, and it will be enforced on remote nodes. Regular UID remapping is a user convenience feature, while root squashing is a security feature.
2. While UID remapping requires having an external infrastructure for mapping between local names and globally unique names, no such infrastructure is necessary for enabling root squashing.

When both UID remapping and root squashing are enabled, root squashing overrides the normal UID remapping mechanism for the root user.

Mounting a file system owned and served by another GPFS cluster

This is an example of how to mount a file system owned and served by another GPFS cluster. OpenSSL must be installed on all nodes in the involved clusters before using these instructions.

See the GPFS Frequently Asked Questions at publib.boulder.ibm.com/infocenter/clresctr/topic/com.ibm.cluster.gpfs.doc/gpfs_faqs/gpfsclustersfaq.html for current OpenSSL version requirements and for information on the supported cipher suites.

The procedure to set up remote file system access involves the generation and exchange of authorization keys between the two clusters. In addition, the administrator of the GPFS cluster that owns the file system needs to authorize the remote clusters that are to access it, while the administrator of the GPFS cluster that seeks access to a remote file system needs to define to GPFS the remote cluster and file system whose access is desired.

In this example, **cluster1** is the name of the cluster that owns and serves the file system to be mounted, and **cluster2** is the name of the cluster that desires to access the file system.

Note: The following example uses AUTHONLY as the authorization setting. When you specify AUTHONLY for authentication, GPFS checks network connection authorization. However, data sent over the connection is not protected.

1. On **cluster1**, the system administrator issues the **mmauth** command to generate a public/private key pair. The key pair is placed in **/var/mmfs/ssl**:

   ```
   mmauth genkey new
   ```

2. On **cluster1**, the system administrator enables authorization by issuing:

   ```
   mmauth update . -l AUTHONLY
   ```

3. The system administrator of **cluster1** now gives the file **/var/mmfs/ssl/id_rsa.pub** to the system administrator of **cluster2**, who desires to access the **cluster1** file systems. This operation requires the two administrators to coordinate their activities, and must occur outside of the GPFS command environment.
4. On **cluster2**, the system administrator issues the **mmauth** command to generate a public/private key pair. The key pair is placed in **/var/mmfs/ssl**:

   ```
   mmauth genkey new
   ```

5. On **cluster2**, the system administrator enables authorization by issuing:

   ```
   mmauth update . -l AUTHONLY
   ```

6. The system administrator of **cluster2** gives file **/var/mmfs/ssl/id_rsa.pub** to the system administrator of **cluster1**. This operation requires the two administrators to coordinate their activities, and must occur outside of the GPFS command environment.
7. On **cluster1**, the system administrator issues the **mmauth add** command to authorize **cluster2** to mount file systems owned by **cluster1** utilizing the key file received from the administrator of **cluster2**:

   ```
   mmauth add cluster2 -k cluster2_id_rsa.pub
   ```

   where:

   cluster2
   Is the real name of **cluster2** as given by the **mmlscluster** command on a node in **cluster2**.

   cluster2_id_rsa.pub
   Is the name of the file obtained from the administrator of **cluster2** in Step 6.
8. On **cluster1**, the system administrator issues the **mmauth grant** command to authorize **cluster2** to mount specific file systems owned by **cluster1**:

   ```
   mmauth grant cluster2 -f /dev/gpfs
   ```

9. On **cluster2**, the system administrator now must define the cluster name, contact nodes and public key for **cluster1**:

   ```
   mmremotecluster add cluster1 -n node1,node2,node3 -k cluster1_id_rsa.pub
   ```

   where:

   cluster1
   Is the real name of **cluster1** as given by the **mmlscluster** command on a node in **cluster1**.

   *node1***,** *node2***, and** *node3*
   Are nodes in **cluster1**. The hostname or IP address that you specify must refer to the communications adapter that is used by GPFS as given by the **mmlscluster** command on a node in **cluster1**.

   cluster1_id_rsa.pub
   Is the name of the file obtained from the administrator of **cluster1** in Step 3.

   This permits the cluster desiring to mount the file system a means to locate the serving cluster and ultimately mount its file systems.
10. On **cluster2**, the system administrator issues one or more **mmremotefs** commands to identify the file systems in **cluster1** that are to be accessed by nodes in **cluster2**:

```
mmremotefs add /dev/mygpfs -f /dev/gpfs -C cluster1 -T /mygpfs
```

where:

/dev/mygpfs
: Is the device name under which the file system will be known in **cluster2**.

/dev/gpfs
: Is the actual device name for the file system in **cluster1**.

cluster1
: Is the real name of **cluster1** as given by the **mmlscluster** command on a node in **cluster1**.

/mygpfs
: Is the local mount point in **cluster2**.

11. On **cluster2**, the command:

```
mmmount /dev/mygpfs
```

then mounts the file system.

Table 2 summarizes the commands that the administrators of the two clusters need to issue so that the nodes in **cluster2** can mount the remote file system **fs1**, owned by **cluster1**, assigning **rfs1** as the local name with a mount point of **/mpl**.

Table 2. Summary of commands to set up cross-cluster file system access.

cluster1	**cluster2**
mmauth genkey new	**mmauth genkey new**
mmauth update . -l AUTHONLY	**mmauth update . -l AUTHONLY**
Exchange public keys (file **/var/mmfs/ssl/id_rsa.pub**)	
mmauth add *cluster2 ...*	**mmremotecluster add** *cluster1 ...*
mmauth grant *cluster2* **-f** fs1 ...	**mmremotefs add** rfs1 **-f** fs1 **-C** cluster1 **-T** /rfs1

Managing remote access to GPFS file systems

This is an example of how to manage remote access to GPFS file systems.

To see a list of all clusters authorized to mount file systems owned by **cluster1**, the administrator of **cluster1** issues this command:

```
mmauth show
```

To authorize a third cluster, say **cluster3**, to access file systems owned by **cluster1**, the administrator of **cluster1** issues this command:

```
mmauth add cluster3 -k cluster3_id_rsa.pub
mmauth grant cluster3 -f /dev/gpfs1
```

To subsequently revoke **cluster3** authorization to access a specific file system **gpfs1** owned by **cluster1**, the administrator of **cluster1** issues this command:

```
mmauth deny cluster3 -f /dev/gpfs1
```

To completely revoke **cluster3** authorization to access file systems owned by **cluster1**, the administrator of **cluster1** issues this command:

```
mmauth delete cluster3
```

Using remote access with public and private IP addresses

GPFS permits the use of both public and private IP address. Private IP addresses are typically used to communicate on private networks.

Private IP addresses are on one of these subnets:

- 10.0.0.0
- 172.16.0.0
- 192.168.0.0

See *The Internet Engineering Task Force* and search on *RFC 1597 - Address Allocation for Private Internets*.

Use the **mmchconfig** command, **subnets** attribute, to specify the private IP addresses to be accessed by GPFS.

Figure 4 on page 8 describes an AIX cluster named **CL1** with nodes named **CL1N1**, **CL1N2**, and so forth, a Linux cluster named **CL2** with nodes named **CL2N1**, **CL2N2**, and another Linux cluster named **CL3** with a node named **CL3N1**. Both Linux clusters have public Ethernet connectivity, and a Gigabit Ethernet configured with private IP addresses (10.200.0.1 through 10.200.0.24), not connected to the public Ethernet. The High Performance Switch on the AIX cluster, **CL1** is configured using public IP addresses on the 7.2.24/13 subnet, and is accessible from the outside.

With the use of both public and private IP addresses for some of the nodes, the setup works as follows:

1. All clusters must be created using host names or public IP addresses corresponding to the public network.
2. Using the **mmchconfig** command for the **CL1** cluster, add the attribute: **subnets=7.2.24.0**.

   This allows all **CL1** nodes to communicate using the IBM High Performance Switch. Remote mounts between **CL2** and **CL1** will use the public Ethernet for TCP/IP communication, since the **CL2** nodes are not on the 7.2.24.0 subnet.
3. GPFS assumes subnet specifications for private networks are independent between clusters (private networks are assumed not physically connected between clusters). The remaining steps show how to indicate that a private network is shared between clusters.
4. Using the **mmchconfig** command for the **CL2** cluster, add the **subnets='10.200.0.0/CL2N*.kgn.ibm.com 10.200.0.0/CL3N*.kgn.ibm.com'** attribute.

   The **10.200.0.0/CL2N*.kgn.ibm.com** portion allows all **CL2** nodes to communicate over their Gigabit Ethernet. The **10.200.0.0/CL3N*.kgn.ibm.com** portion allows remote mounts between clusters **CL2** and **CL3** to communicate over their Gigabit Ethernet.

   The **subnets='10.200.0.0/CL3N*.kgn.ibm.com 10.200.0.0/CL2N*.kgn.ibm.com'** attribute indicates that the private 10.200.0.0 network extends to all nodes in clusters that have a name beginning with **CL2N** and **CL3N**. This way, any two nodes in the **CL2** and **CL3** cluster can communicate through the Gigabit Ethernet.
5. Using the **mmchconfig** command for the **CL3** cluster, add the **subnets='10.200.0.0/CL3N*.kgn.ibm.com 10.200.0.0/CL2N*.kgn.ibm.com'** attribute.

   The **10.200.0.0/CL3N*.kgn.ibm.com** portion allows all **CL3** nodes to communicate over their Gigabit Ethernet. The **10.200.0.0/CL2N*.kgn.ibm.com** portion allows remote mounts between clusters **CL3** and **CL2** to communicate over their Gigabit Ethernet.

   The **subnets='10.200.0.0/CL3N*.kgn.ibm.com 10.200.0.0/CL2N*.kgn.ibm.com'** attribute indicates that the private 10.200.0.0 network extends to all nodes in clusters that have a name beginning with **CL2N** and **CL3N**. This way, any two nodes in the **CL2** and **CL3** cluster can communicate through the Gigabit Ethernet.

Use the **subnets** attribute of the **mmchconfig** command when you wish the GPFS cluster to leverage additional, higher performance network connections that are available to the nodes in the cluster, or between clusters.

Note: Use of the **subnets** attribute does not ensure a highly available system. If the GPFS daemon is using the IP address specified by the **subnets** attribute, and that interface goes down, GPFS does not switch to the other network.

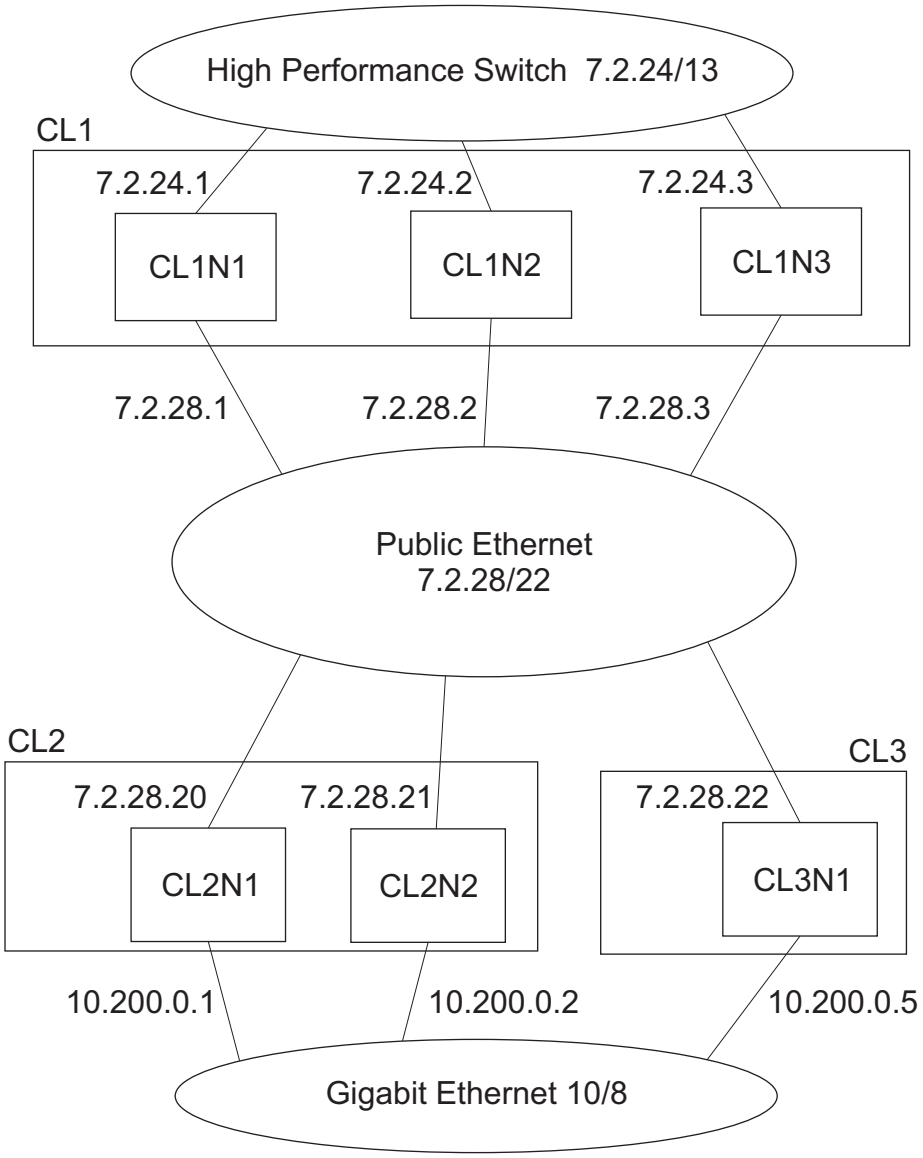


Figure 4. Use of public and private IP addresses in three GPFS clusters

Using multiple security levels

A cluster that owns a file system whose access is to be permitted from other clusters, can designate a different security level for each connecting cluster.

When multiple security levels are specified, the following rule applies: each connection uses the security level of the connecting node, unless that security level is **AUTHONLY**. In this case, the security level of the node accepting the connection is used instead. This means that a connection will use **AUTHONLY** if and only if both nodes exist in clusters that are required to use security method **AUTHONLY**.

To specify a different security level for different clusters requesting access to a given cluster, use the **mmauth -l** *cipherList* command. Several examples follow to illustrate:

1. In this example, **cluster1** and **cluster2** are located on the same trusted network, and **cluster3** is connected to both of them with an untrusted network. The system administrator chooses these security levels:
   - A **cipherList** of **AUTHONLY** for connections between **cluster1** and **cluster2**
   - A **cipherList** of **NULL-SHA** for connections between **cluster1** and **cluster3**
   - A **cipherList** of **NULL-SHA** for connections between **cluster2** and **cluster3**

   The administrator of **cluster1** issues these commands:

   ```
   mmauth add cluster2 -k keyFile -l AUTHONLY
   mmauth add cluster3 -k keyFile -l NULL-SHA
   ```

2. In this example, **cluster2** is accessing file systems owned by **cluster1** using a **cipherList** of **AUTHONLY**, but the administrator of **cluster1** has decided to require to a more secure **cipherList**. The administrator of **cluster1** issues this command:

   ```
   mmauth update cluster2 -l NULL-SHA
   ```

   Existing connections will be upgraded from **AUTHONLY** to **NULL-SHA**.

Changing security keys

When working with GPFS file systems accessed by other GPFS clusters, it might be necessary to generate a new public/private access key. This can be done without disturbing existing connections, provided the following procedure is followed.

To accomplish this, the cluster that owns and serves the file system is made to temporarily have two access keys (referred to as the 'old key' and the 'new key'), which are both valid at the same time. The clusters currently accessing the file system can then change from the old key to the new key without interruption of file system access.

In this example, **cluster1** is the name of the cluster that owns and serves a file system, and **cluster2** is the name of the cluster that has already obtained access to this file system, and is currently using it. Here, the system administrator of **cluster1** changes the access key without severing the connection obtained by **cluster2**.

1. On **cluster1**, the system administrator issues the **mmauth genkey new** command to generate a new public/private access key pair. The key pair is placed in **/var/mmfs/ssl**:

   ```
   mmauth genkey new
   ```

   After this command is issued, **cluster1** will have two keys (referred to as the 'old key' and the 'new key') that both may be used to access **cluster1** file systems.
2. The system administrator of **cluster1** now gives the file **/var/mmfs/ssl/id_rsa.pub** (that contains the new key) to the system administrator of **cluster2**, who desires to continue to access the **cluster1** file systems. This operation requires the two administrators to coordinate their activities, and must occur outside of the GPFS command environment.
3. On **cluster2**, the system administrator issues the **mmremotecluster update** command to make the new key known to his system:

   ```
   mmremotecluster update cluster1 -k cluster1_id_rsa.pub
   ```

   where:

   cluster1
   : Is the real name of **cluster1** as given by the **mmlscluster** command on a node in **cluster1**.

   cluster1_id_rsa.pub
   : Is the name of the file obtained from the administrator of **cluster1** in Step 2.

This permits the cluster desiring to mount the file system to continue mounting file systems owned by **cluster1**.

4. On **cluster1**, the system administrator verifies that all clusters desiring to access **cluster1** file systems have received the new key and activated it using the **mmremotecluster update** command.
5. On **cluster1**, the system administrator issues the **mmauth genkey commit** command to commit the new key as the only valid access key. The old key will no longer be accepted once this command completes successfully:

   ```
   mmauth genkey commit
   ```

   Once the new public key has been committed, the old public key will no longer be accepted. As a result, any remote cluster administrator who has not been given the new key (Step 2 on page 9 above) and run **mmremotecluster update** (Step 3 on page 9 above) will no longer be able to mount file systems owned by **cluster1**.

Similarly, the administrator of **cluster2** may decide to change the access key for **cluster2**:

1. On **cluster2**, the system administrator issues the **mmauth genkey new** command to generate a new public/private access key pair. The key pair is placed in **/var/mmfs/ssl**:

   ```
   mmauth genkey new
   ```

   After this command is issued, **cluster2** will have two keys (referred to as the 'old key' and the 'new key') that both may be used when a connection is established to any of the nodes in **cluster2**.
2. The system administrator of **cluster2** now gives the file **/var/mmfs/ssl/id_rsa.pub** (that contains the new key) to the system administrator of **cluster1**, the owner of the file systems. This operation requires the two administrators to coordinate their activities, and must occur outside of the GPFS command environment.
3. On **cluster1**, the system administrator issues the **mmauth update** command to make the new key known to his system:

   ```
   mmauth update cluster2 -k cluster2_id_rsa.pub
   ```

   where:

   cluster2
   Is the real name of **cluster2** as given by the **mmlscluster** command on a node in **cluster2**.

   cluster2_id_rsa.pub
   Is the name of the file obtained from the administrator of **cluster2** in Step 2.

   This permits the cluster desiring to mount the file system to continue mounting file systems owned by **cluster1**.
4. The system administrator of **cluster2** verifies that the administrator of **cluster1** has received the new key and activated it using the **mmauth update** command.
5. On **cluster2**, the system administrator issues the **mmauth genkey commit** command to commit the new key as the only valid access key. The old key will no longer be accepted once this command completes successfully:

   ```
   mmauth genkey commit
   ```

Additional information about GPFS file systems accessed by nodes that belong to other GPFS clusters

There is some additional information about this topic that you should take into consideration.

When working with GPFS file systems accessed by nodes that belong to other GPFS clusters, consider the following points:

1. A file system is administered only by the cluster where the file system was created. Other clusters may be allowed to mount the file system, but their administrators cannot add or delete disks, change

characteristics of the file system, enable or disable quotas, run the **mmfsck** command, and so forth. The only commands that other clusters can issue are list type commands, such as: **mmlsfs**, **mmlsdisk**, **mmlsmount**, and **mmdf**.

2. Since each cluster is managed independently, there is no automatic coordination and propagation of changes between clusters, like there is between the nodes within a cluster.

   This means that if the administrator of **cluster1** (the owner of file system **gpfs1**) decides to delete it or rename it, the information for **gpfs1** in **cluster2** becomes obsolete, and an attempt to mount **gpfs1** from **cluster2** will fail. It is assumed that when such changes take place, the two administrators will inform each other. The administrator of **cluster2** can then use the **update** or **delete** options of the **mmremotefs** command to make the appropriate changes.
3. Similar to the item above, if the names of the contact nodes change, the name of the cluster changes, or the public key file changes, use the **update** option of the **mmremotecluster** command to reflect the changes.
4. Use the **show** option of the **mmremotecluster** and **mmremotefs** commands to display the current information about remote clusters and file systems.
5. If the cluster that owns a file system has a **maxblocksize** configuration parameter that is different from the **maxblocksize** configuration parameter of the cluster that desires to mount a file system, a mismatch may occur and file system mount requests may fail with messages to this effect. Check your **maxblocksize** configuration parameters on both clusters using the **mmlsconfig** command. Correct any discrepancies with the **mmchconfig** command.

Chapter 2. Policy-based data management for GPFS

GPFS can help you achieve Information Lifecycle Management (ILM) efficiencies through powerful policy-driven automated tiered storage management. The GPFS ILM toolkit helps you manage sets of files, pools of storage and automate the management of file data.

Using these tools, GPFS can automatically determine where to physically store your data regardless of its placement in the logical directory structure. Storage pools, filesets and user-defined policies provide the ability to match the cost of your storage resources to the value of your data.

GPFS policy-based ILM tools allow you to:

- Create *storage pools* to provide a way to partition a file system's storage into collections of disks or a redundant array of independent disks (RAIDs) with similar properties that are managed together as a group. GPFS has three types of storage pools:
  - A required **system** storage pool that you create and manage through GPFS
  - Optional user storage pools that you create and manage through GPFS
  - Optional external storage pools that you define with GPFS policy rules and manage through an external application such as Tivoli® Storage Manager
- Create *filesets* to provide a way to partition the file system namespace to allow administrative operations at a finer granularity than that of the entire file system. See "Filesets" on page 18.
- Create *policy rules* based on data attributes to determine initial file data placement and manage file data placement throughout the life of the file. See "Policies and rules" on page 23.

Storage pools

Physically, a *storage pool* is a collection of disks or RAID arrays. Storage pools also allow you to group multiple storage systems within a file system.

Using storage pools, you can create tiers of storage by grouping storage devices based on performance, locality, or reliability characteristics. For example, one pool could be an enterprise class storage system that hosts high-performance fibre-channel disks and another pool might consist of numerous disk controllers that host a large set of economical SATA disks.

There are two types of storage pools in GPFS, internal storage pools and external storage pools. Internal storage pools are managed within GPFS. External storage pools are managed by an external application such as Tivoli Storage Manager. For external storage pools, GPFS provides tools that allow you to define an interface that your external storage manager uses to access your data. GPFS does not manage the data placed in external storage pools. Instead, GPFS manages the movement of data to and from external storage pools. Storage pools allow you to perform complex operations such as moving, mirroring, or deleting files across multiple storage devices, providing storage virtualization and a single management context.

Internal GPFS storage pools are meant for managing online storage resources. External storage pools are intended for use as near-line storage and for archival and backup operations. However, both types of storage pools provide you with a method to partition file system storage for considerations such as:

- Improved price-performance by matching the cost of storage to the value of the data
- Improved performance by:
  - Reducing the contention for premium storage
  - Reducing the impact of slower devices
  - Allowing you to retrieve archived data when needed
- Improved reliability by providing for:

- Replication based on need
- Better failure containment
- Creation of new storage pools as needed

For additional information, refer to:

- “Internal GPFS storage pools”
- “External storage pools” on page 17

Internal GPFS storage pools

The internal GPFS storage pool to which a disk belongs is specified as an attribute of the disk in the GPFS cluster. You specify the disk attributes as a field in each disk descriptor when you create the file system or when adding disks to an existing file system. GPFS allows a maximum of eight internal storage pools per file system. One of these storage pools is the required **system** storage pool. The other seven internal storage pools are optional user storage pools.

GPFS assigns file data to internal storage pools:

- When they are initially created; the storage pool is determined by the file placement policies that are in effect when the file is created.
- When the attributes of the file, such as file size or access time, match the rules of an active policy that directs GPFS to migrate the data to a different storage pool.

For additional information, refer to:

- “The system storage pool”
- “User storage pools”
- “Managing storage pools” on page 15

The system storage pool

The **system** storage pool contains file system control structures, reserved files, directories, symbolic links, special devices, as well as the metadata associated with regular files, including indirect blocks, extended attributes, and so forth. The **system** storage pool can also contain user data. There is only one **system** storage pool per file system, and it is automatically created when the file system is created. File systems created prior to GPFS 3.1 have all of their disks belonging to the **system** storage pool.

Important: It is recommended that you use highly-reliable disks for the **system** storage pool so that the cluster always has a copy of the system metadata.

The amount of metadata grows as you add files to the system. Therefore, it is recommended that you monitor the **system** storage pool to ensure that there is always enough space to accommodate growth. The **system** storage pool typically requires a small percentage of the total storage capacity that GPFS manages. However, the percentage required by the **system** storage pool varies depending on your environment. You can monitor the amount of space available in the **system** storage pool with the **mmdf** command. If the available space in the system storage pool begins to run low, you can increase the available space by purging files or adding disks to the system storage pool.

User storage pools

All user data for a file is stored in the assigned storage pool as determined by your file placement rules. In addition, file data can be migrated to a different storage pool according to your file management policies. For more information on policies, see “Policies and rules” on page 23.

A user storage pool contains the blocks of data that make up user files. GPFS stores the data that describes the files, called file metadata, separately from the actual file data in the **system** storage pool. You can create one or more user storage pools, and then create policy rules to indicate where the data blocks for a file should be stored.

Managing storage pools

Managing your storage pools includes:

Creating storage pools:

The storage pool that a disk belongs to is an attribute of each disk and is specified as a field in each disk descriptor when the file system is created using the **mmcrfs** command or when disks are added to an existing file system with the **mmadddisk** command. Adding a disk with a new storage pool name in the disk descriptor automatically creates the storage pool.

Storage pool names:

- Must be unique within a file system, but not across file systems.
- Cannot be larger than 255 alphanumeric characters.
- Are case sensitive. **MYpool** and **myPool** are distinct storage pools.

The storage pool field is the seventh field on a disk descriptor:

```
DiskName:::DiskUsage:FailureGroup::StoragePool:
```

If a storage pool is not specified in the disk descriptor, the disk is by default assigned to the **system** storage pool.

Changing the storage pool assignment for a disk:

Once a disk is assigned to a storage pool, the pool assignment cannot be changed using either the **mmchdisk** command or the **mmrpldisk** command. To move a disk to another pool:

1. Delete the disk from its current pool by issuing the **mmdeldisk** command. This will move the data to the remaining disks in the storage pool.
2. Add the disk to the new pool by issuing the **mmadddisk** command.
3. Rebalance the data across all disks in the new storage pool by issuing the **mmrestripefs -P** command.

Changing the storage pool assignment for a file:

A root user can change the storage pool that a file is assigned to by either:

- Running **mmapplypolicy** with an appropriate set of policy rules.
- Issuing the **mmchattr -P** command.

By default, both of these commands migrate data immediately (this is the same as using the **-I yes** option for these commands). If desired, you can delay migrating the data by specifying the **-I defer** option for either command. Using the defer option, the existing data does not get moved to the new storage pool until either the **mmrestripefs** command or the **mmrestripefile** command are executed. For additional information, refer to:

- “Policies” on page 23
- “Rebalancing files in a storage pool” on page 17

Deleting storage pools:

Deleting the **system** storage pool is not allowed. In order to delete the **system** storage pool, you must delete the file system.

In order to delete a user storage pool, you must delete all its disks using the **mmdeldisk** command. When GPFS deletes the last remaining disk from a user storage pool, the storage pool is also deleted. To delete a storage pool, it must be completely empty. A migration policy along with the **mmapplypolicy** command could be used to do this.

Listing storage pools for a file system:

To list the storage pools available for a specific file system, issue the **mmlsfs -P** command.

For example, this command

```
mmlsfs fs1 -P
```

produces output similar to this:

```
flag value          description
---- -------------- -----------------------------------------------------
 -P  system;sp1;sp2  Disk storage pools in file system
```

For file system **fs1**, there are three storage pools: the **system** storage pool and user storage pools named **sp1** and **sp2**.

Listing storage pools for a file:

To display the assigned storage pool and the name of the fileset that includes the file, issue the **mmlsattr -L** command.

For example, this command:

```
mmlsattr -L myfile
```

produces output similar to this:

```
file name:            myfile
metadata replication: 2 max 2
data replication:     1 max 2
flags:
storage pool name:    sp1
fileset name:         root
snapshot name:
```

File **myfile** is assigned to the storage pool named **sp1** and is part of the root fileset.

Listing disks in a storage pool and associated statistics:

To list the disks belonging to a storage pool, issue the **mmdf -P** command.

For example, this command:

```
mmdf fs1 -P sp1
```

produces output similar to this:

```
disk                disk size  failure holds    holds              free KB             free KB
name                    in KB    group metadata data        in full blocks        in fragments
--------------- ------------- -------- -------- ----- -------------------- -------------------
Disks in storage pool: sp1
vp4vsdn05            17760256        6 no       yes        11310080 ( 64%)        205200 ( 1%)
vp5vsdn05            17760256        6 no       yes        11311104 ( 64%)        205136 ( 1%)
vp6vsdn05            17760256        6 no       yes        11300352 ( 64%)        206816 ( 1%)
vp7vsdn05            17760256        6 no       yes        11296256 ( 64%)        209872 ( 1%)
vp0vsdn05            17760256        6 no       yes        11293696 ( 64%)        207968 ( 1%)
vp1vsdn05            17760256        6 no       yes        11293184 ( 64%)        206464 ( 1%)
vp2vsdn05            17760256        6 no       yes        11309056 ( 64%)        203248 ( 1%)
vp3vsdn05            17760256        6 no       yes        11269120 ( 63%)        211456 ( 1%)
             -------------                         -------------------- -------------------
(pool total)        142082048                              90382848 ( 64%)       1656160 ( 1%)
```

This example shows that storage pool **sp1** in file system **fs1** consists of eight disks and identifies details for each disk including:

- Name
- Size
- Failure group
- Data type
- Free space

Rebalancing files in a storage pool:

A root user can rebalance file data across all disks in a file system by issuing the **mmrestripefs** command. Optionally:

- Specifying the **-P** option rebalances only those files assigned to the specified storage pool.
- Specifying the **-p** option rebalances the file placement within the storage pool. Files assigned to one storage pool, but with data in a different pool, (referred to as ill-placed files), have their data migrated to the correct pool.

Using replication with storage pools: To enable data replication in a storage pool, you must make certain that there are at least two failure groups within the storage pool. This is necessary because GPFS maintains separation between storage pools and performs file replication within each storage pool. In other words, a file and its replica must be in the same storage pool. This also means that if you are going to replicate the entire file system, every storage pool in the file system must have at least two failure groups.

Note: Depending on the configuration of your file system, if you try to enable file replication in a storage pool having only one failure group, GPFS will either give you a warning or an error message.

External storage pools

When you initially create a file, GPFS assigns that file to an internal storage pool. Internal storage pools support various types of online storage. To move data from online storage to offline or near-line storage, you can use external storage pools. External storage pools use a flexible interface driven by GPFS policy rules that simplify data migration to and from other types of storage such as tape storage. For additional information, refer to “Policies and rules” on page 23.

You can define multiple external storage pools at any time using GPFS policy rules. To move data to an external storage pool, the GPFS policy engine evaluates the rules that determine which files qualify for transfer to the external pool. From that information, GPFS provides a list of candidate files and executes the script specified in the rule that defines the external pool. That executable script is the interface to the external application, such as Tivoli Storage Manager (TSM), that does the actual migration of data into an external pool. Using the external pool interface, GPFS gives you the ability to manage information by allowing you to:

1. Move files and their extended attributes onto low-cost near-line storage when demand for the files diminishes.
2. Restore the files, with all of their previous access information, onto online storage whenever the files are needed.

External pool requirements

With external pools, GPFS provides metadata processing and the flexibility of using extended file attributes. The external storage manager is responsible for moving files from GPFS and returning them upon the request of an application accessing the file system. Therefore, when you are using external storage pools, you must use an external file management application such as TSM. The external application is responsible for maintaining the file once it has left the GPFS file system. For example, GPFS policy rules create a list of files that are eligible for migration. GPFS hands that list to TSM which migrates the files to tape and creates a reference file in the file system that has pointers to the tape image. When a file is requested, it is automatically retrieved from the external storage pool and placed back in an internal storage pool. As an alternative, you can use a GPFS policy rule to retrieve the data in advance of a user request.

The number of external storage pools is only limited by the capabilities of your external application. GPFS allows you to define external storage pools at any time by writing a policy that defines the pool and makes that location known to GPFS. External storage pools are defined by policy rules and initiated by either storage thresholds or use of the **mmapplypolicy** command.

For additional information, refer to “Working with external storage pools” on page 37.

Filesets

In most file systems, a file hierarchy is represented as a series of directories that form a tree-like structure. Each directory contains other directories, files, or other file-system objects such as symbolic links and hard links. Every file system object has a name associated with it, and is represented in the namespace as a node of the tree.

In addition, GPFS utilizes a file system object called a *fileset*. A fileset is a subtree of a file system namespace that in many respects behaves like an independent file system. Filesets provide a means of partitioning the file system to allow administrative operations at a finer granularity than the entire file system:

- Filesets can be used to define quotas on both data blocks and inodes.
- Filesets are the smallest unit defined for a snapshot, allowing an administrator to save, restore, and define backup procedures for filesets independently.
- The owning fileset is an attribute of each file and can be specified in a policy to control initial data placement, migration, and replication of the file’s data. See “Policies and rules” on page 23.
- The maximum number of filesets that GPFS supports is 1000 filesets per file system.

When the file system is created, only one fileset, called the *root* fileset, exists. It contains the root directory as well as any system files such as quota files. As new files and directories are created, they automatically become part of the parent directory’s fileset. The fileset to which a file belongs is largely transparent for ordinary file access, but the containing fileset can be displayed along with the other attributes of each file using the **mmlsattr -L** command.

When upgrading an existing file system to versions of GPFS that support filesets (GPFS 3.1 or later), all existing files and directories are assigned to the root fileset.

Fileset namespace

A newly created fileset consists of an empty directory for the root of the fileset, and it is initially not linked into the file system's namespace. A newly created fileset is not visible to the user until it is attached to the namespace by issuing the **mmlinkfileset** command. Filesets are attached to the namespace with a special link called a *junction*. A junction is a special directory entry, much like a POSIX hard link, that connects a name in a directory of one fileset to the root directory of another fileset. Only one junction is allowed per fileset, so that a fileset has a unique position in the namespace and a unique path to any of its directories. The target of the junction is referred to as the *child fileset*, and a fileset can have any number of children. From the user's viewpoint, a junction always appears as if it were a directory, but the user is not allowed to issue the **unlink** or **rmdir** commands on a junction.

Once a fileset has been created and linked into the namespace, an administrator can unlink the fileset from the namespace by issuing the **mmunlinkfileset** command. This makes all files and directories within the fileset inaccessible. If other filesets were linked below it, the other filesets become inaccessible, but they do remain linked and will become accessible again when the fileset is re-linked. Unlinking a fileset, like unmounting a file system, fails if there are open files. The **mmunlinkfileset** command has a force option to close the files and force the unlink. Once this is done, references to these files will result in **ESTALE** errors. Once a fileset is unlinked, it can be re-linked into the namespace at its original location or any other location (it cannot be linked into its children since they are not part of the namespace while the parent fileset is unlinked).

The namespace inside a fileset is restricted to a single, connected subtree. In other words, a fileset has only one root directory and no other entry points such as hard links from directories in other filesets. Filesets are always connected at the root directory and only the junction makes this connection. Consequently, hard links cannot cross fileset boundaries. Symbolic links, of course, can be used to provide shortcuts to any file system object in the namespace.

Note: The root directory of a GPFS file system is also the root of the root fileset. The root fileset is an exception. The root fileset is attached to the local namespace using the standard **mount** command. It cannot be created, linked, unlinked or deleted using the GPFS fileset commands.

See "Managing filesets" on page 21.

Filesets and quotas

The GPFS quota commands support the **-j** option for fileset block and inode allocation. The quota limit on blocks and inodes in a fileset are independent of the limits for specific users or groups of users. See these commands:

- **mmdefedquota**
- **mmdefedquotaon**
- **mmdefedquotaoff**
- **mmedquota**
- **mmlsquota**
- **mmquotaoff**
- **mmrepquota**

Filesets and storage pools

Filesets are not specifically related to storage pools, although each file in a fileset physically resides in blocks in a storage pool. This relationship is many-to-many; each file in the fileset can be stored in a different user storage pool. A storage pool can contain files from many filesets. However, all of the data for a particular file is wholly contained within one storage pool.

Using file-placement policies, you can specify that all files created in a particular fileset are to be stored in a specific storage pool. Using file-management policies, you can define how files in a specific fileset are to be moved or deleted during the file's life cycle. See "Policy rule syntax definitions" on page 25.

Filesets and snapshots

A GPFS snapshot preserves the content of the entire file system, including all its filesets, even unlinked ones. The state of filesets in the snapshot is unaffected by changes made to filesets in the active file system, such as unlink, link or delete. The saved file system can be accessed through the **.snapshots** directories and the namespace, including all linked filesets, appears as it did when the snapshot was created. Unlinked filesets are inaccessible in the snapshot, as they were in the active file system. However, restoring a snapshot also restores the unlinked filesets, which can then be re-linked and accessed.

If a fileset is included in a snapshot, it can be deleted but it is not entirely removed from the file system. In this case, the fileset is emptied of all contents and given a status of 'deleted'. The contents of a fileset remain available in the snapshots that include the fileset (that is, through some path containing a **.snapshots** component) even after the fileset is deleted, since all the contents of the fileset are saved when a snapshot is created. The fileset remains in the deleted state until the last snapshot containing it is deleted, at which time the fileset is automatically deleted.

A fileset is included in a snapshot if the snapshot is created after the fileset was created. Deleted filesets appear in the output of the **mmlsfileset** command, and the **-L** option can be used to display the latest snapshot that includes a fileset.

The filesets included in the snapshot are restored to their former state, and newer filesets are deleted. In particular, restoring a snapshot may undelete deleted filesets and change linked filesets to unlinked or vice versa. As with other changes made to the active file system by the restore operation, it also does not affect the fileset state saved in other snapshots. This means that if the name of a fileset was changed since the snapshot was taken, the old fileset name will be restored.

Filesets and backup

The **mmbackup** command and TSM are unaware of the existence of filesets. When restoring a file system that had been backed up to TSM, the files are restored to their original path names, regardless of the filesets of which they were originally a part.

TSM has no mechanism to create or link filesets during restore. Therefore, if a file system is migrated to TSM and then filesets are unlinked or deleted, restore or recall of the file system does not restore the filesets.

During a full restore from backup, all fileset information is lost and all files are restored into the root fileset. It is recommended that you save the output of the **mmlsfileset** command to aid in the reconstruction of fileset names and junction locations. Saving **mmlsfileset -L** also allows reconstruction of fileset comments. Both command outputs are needed to fully restore the fileset configuration.

A partial restore can also lead to confusion if filesets have been deleted, unlinked, or their junctions moved, since the backup was made. For example, if the backed up data was in a fileset that has since been unlinked, the restore process puts it into files and directories in the parent fileset. The unlinked fileset cannot be re-linked into the same location until the restored data is moved out of the way. Similarly, if the fileset was deleted, restoring its contents does not recreate the deleted fileset, but the contents are instead restored into the parent fileset.

Since the **mmbackup** command operates by traversing the directory structure, it does not include the contents of unlinked filesets, even though they are part of the file system. If it is desired to include these filesets in the backup, they should be re-linked, perhaps into a temporary location. Conversely, temporarily unlinking a fileset is a convenient mechanism to exclude it from a backup.

In summary, fileset information should be saved by periodically recording **mmlsfileset** output somewhere in the file system, where it is preserved as part of the backup process. During restore, care should be exercised when changes in the fileset structure have occurred since the backup was created.

Attention: If you are using the TSM Backup Archive client you must use caution when you unlink filesets that contain data backed up by TSM. TSM tracks files by pathname and does not track filesets. As a result, when you unlink a fileset, it appears to TSM that you deleted the contents of the fileset. Therefore, the TSM Backup Archive client inactivates the data on the TSM server which may result in the loss of backup data during the expiration process.

Managing filesets

Managing your filesets includes:

Creating a fileset

The system administrator creates a new fileset by issuing the **mmcrfileset** command. A newly created fileset consists of an empty directory for the root of the fileset and it is initially not linked into the existing namespace. Consequently, a new fileset is not visible, nor can files be added to it, but the fileset name is valid and the administrator can establish quotas on it, or policies for it. The administrator must link the fileset into its desired location in the file system's name space by issuing the **mmlinkfileset** command in order to make use of it.

The fileset can be linked anywhere in the namespace. It can be linked to the root directory or any subdirectory, and to the root fileset or to any other fileset. The fileset can be linked into only one location. Finally, the administrator can change the ownership and permissions for the new fileset's root directory, which default to **root** and 0700, to allow users access to it. Files and directories copied into, or created within, the fileset's directory will become part of the new fileset.

Fileset names must follow these conventions:

- Are character strings and must be less than 256 characters in length.
- Must be unique within a file system.
- The name **root** is reserved for the fileset of the files system's root directory.

Deleting a fileset

Filesets are deleted with the **mmdelfileset** command. The command fails if the fileset is currently linked into the namespace. By default, the **mmdelfileset** command fails if the fileset contains any contents except for an empty root directory. The force (**-f**) option can be used to delete all contents from the fileset, unlink any child filesets, then delete the fileset.

Note: The root fileset **cannot** be deleted.

Linking a fileset

After the fileset is created, a junction must be created to link it to the desired location in the file system's namespace using the **mmlinkfileset** command. The fileset can be linked into only one location anywhere in the namespace, specified by the *JunctionPath* parameter:

- The root directory
- Any subdirectory
- The root fileset or to any other fileset

If *JunctionPath* is not specified, the junction is created in the current directory and has the same name as the fileset being linked. After the command completes, the new junction appears as an ordinary directory, except that the user is not allowed to unlink or delete it with the **rmdir** command it. The user can use the **mv** command on the directory to move to a new location in the parent fileset, but the **mv** command is not allowed to move the junction to a different fileset.

Unlinking a fileset

A junction to a fileset is removed by issuing the **mmunlinkfileset** command. The unlink fails if there are files open in the fileset. The **mmunlinkfileset** command unlinks the fileset only from the active directory namespace. After issuing the **mmunlinkfileset** command, the fileset can be re-linked to a different parent using the **mmlinkfileset** command. Until the fileset is re-linked, it is not accessible.

Note: The root fileset **cannot** be unlinked.

Attention: If you are using the TSM Backup Archive client you must use caution when you unlink filesets that contain data backed up by TSM. TSM tracks files by pathname and does not track filesets. As a result, when you unlink a fileset, it appears to TSM that you deleted the contents of the fileset. Therefore, the TSM Backup Archive client inactivates the data on the TSM server which may result in the loss of backup data during the expiration process.

Changing fileset attributes

To change a filesets's junction, you have to first unlink the fileset using the **mmunlinkfileset** command, and then create the new junction using the **mmlinkfileset** command.

To change the name of a fileset, or the comment associated with the fileset, use the **mmchfileset** command.

Displaying fileset information

Fileset status and attributes are displayed with the **mmlsfileset** command. The attributes displayed include:

- Name of the fileset.
- Fileset identifier of the fileset.
- Junction path to the fileset.
- Status of the fileset.
- Root inode number of the fileset.
- Path to the fileset (if linked).
- User provided comments (if any).
- Other attributes. See **mmlsfileset** for a complete list.

To display the name of the fileset that includes a given file, issue the **mmlsattr** command and specify the **-L** option.

Policies and rules

GPFS provides a means to automate the management of files using policies and rules. Properly managing your files allows you to efficiently use and balance your premium and less expensive storage resources.

GPFS supports these policies:

- *File placement policies* are used to automatically place newly created files in a specific storage pool.
- *File management policies* are used to manage files during their lifecycle by moving them to another storage pool, moving them to near-line storage, copying them to archival storage, changing their replication status, or deleting them.

Policies

A *policy* is a set of rules that describes the life cycle of user data based on the file's attributes. Each rule defines an operation or definition, such as migrate to a pool and replicate the file. There are three uses for rules:

- Initial file placement
- File management
- Restoring file data

When a file is created or restored, the placement policy determines the location of the file's data and assigns the file to a storage pool. All data written to that file will be placed in the assigned storage pool.

The placement policy defining the initial placement of newly created files and the rules for placement of restored data must be installed into GPFS with the **mmchpolicy** command. If a GPFS file system does not have a placement policy installed, all the data will be stored into the **system** storage pool. Only one placement policy can be installed at a time. If you switch from one placement policy to another, or make changes to a placement policy, that action has no effect on existing files. However, newly created files are always placed according to the currently installed placement policy.

The management policy determines file management operations such as migration and deletion.

In order to migrate or delete data, you must use the **mmapplypolicy** command. You can define the file management rules and install them in the file system together with the placement rules. As an alternative, you may define these rules in a separate file and explicitly provide them to **mmapplypolicy** using the **-P** option. In either case, policy rules for placement or migration may be intermixed. Over the life of the file, data can be migrated to a different storage pool any number of times, and files can be deleted or restored.

File management rules can also be used to control the space utilization of GPFS online storage pools. When the utilization for an online pool exceeds the specified high threshold value, the GPFS daemon will trigger an event to automatically start **mmapplypolicy** and reduce the utilization of the pool.

GPFS performs error checking for file-placement policies in the following phases:

- When you install a new policy, GPFS checks the basic syntax of all the rules in the policy.
- GPFS also checks all references to storage pools. If a rule in the policy refers to a storage pool that does not exist, the policy is not installed and an error is returned.
- When a new file is created, the rules in the active policy are evaluated in order. If an error is detected, GPFS logs an error, skips all subsequent rules, and returns an **EINVAL** error code to the application.
- Otherwise, the first applicable rule is used to store the file data.

Default file-placement policy:

When a GPFS file system is first created, the default file-placement policy is to assign all files to the **system** storage pool.

For more information on using GPFS commands to manage policies, see "Managing policies" on page 36.

Policy rules

A *policy rule* is an SQL-like statement that tells GPFS what to do with the data for a file in a specific storage pool if the file meets specific criteria. A rule can apply to any file being created or only to files being created within a specific fileset or group of filesets.

Rules specify conditions that, when true, cause the rule to be applied. Conditions that cause GPFS to apply a rule include:

- Date and time when the rule is evaluated, that is, the current date and time
- Date and time when the file was last accessed
- Date and time when the file was last modified
- Fileset name
- File name or extension
- File size
- User ID and group ID

There are eight types of policy rules that allow you to define specific actions that GPFS will implement on the file data. Each rule has clauses that control candidate selection, namely when the rules is allowed to match a file, what files it will match, the order to operate on the matching files and additional attributes to show for each candidate file. Different clauses are permitted on different rules based upon the semantics of the rule.

The rules and their respective syntax diagrams are:

- File placement rule

```
RULE ['RuleName']
  SET POOL 'PoolName'
     [LIMIT (OccupancyPercentage)]
     [REPLICATE (DataReplication)]
     [FOR FILESET (FilesetName[,FilesetName]...)]
     [WHERE SqlExpression]
```

- File migration rule

```
RULE ['RuleName'] [WHEN TimeBooleanExpression]
  MIGRATE
    [FROM POOL 'FromPoolName'
      [THRESHOLD (HighPercentage[,LowPercentage[,PremigratePercentage]])]]
    [WEIGHT (WeightExpression)]
  TO POOL 'ToPoolName'
    [LIMIT (OccupancyPercentage)]
    [REPLICATE (DataReplication)]
    [FOR FILESET (FilesetName[,FilesetName]...)]
    [SHOW (['String'] SqlExpression)]
    [WHERE SqlExpression]
```

- File deletion rule

```
RULE ['RuleName'] [WHEN TimeBooleanExpression]
  DELETE
    [FROM POOL 'FromPoolName'
      [THRESHOLD (HighPercentage[,LowPercentage])]]
    [WEIGHT (WeightExpression)]
    [FOR FILESET (FilesetName[,FilesetName]...)]
    [SHOW (['String'] SqlExpression)]
    [WHERE SqlExpression]
```

- File exclusion rule

```
RULE ['RuleName'] [WHEN TimeBooleanExpression]
  EXCLUDE
   [FROM POOL 'FromPoolName']
   [FOR FILESET (FilesetName[,FilesetName]...)]
   [WHERE SqlExpression]
```

- File list rule

```
RULE ['RuleName'] [WHEN TimeBooleanExpression]
  LIST 'ListName'
    [DIRECTORIES_PLUS]
    [EXCLUDE]
    [FROM POOL 'FromPoolName'
      [THRESHOLD (HighPercentage[,LowPercentage])]]
    [WEIGHT (WeightExpression)]
    [FOR FILESET (FilesetName[,FilesetName]...)]
    [SHOW (['String'] SqlExpression)]
    [WHERE SqlExpression]
```

- File restore rule

```
RULE ['RuleName']
  RESTORE TO POOL 'PoolName'
    [LIMIT (OccupancyPercentage)]
    [REPLICATE (DataReplication)]
    [FOR FILESET (FilesetName[,FilesetName]...)]
    [WHERE SqlExpression]
```

- External storage pool definition rule

```
RULE ['RuleName']
  EXTERNAL POOL 'PoolName'
   EXEC 'InterfaceScript'
    [OPTS 'OptionsString ...']
```

- External list definition rule

```
RULE ['RuleName']
  EXTERNAL LIST 'ListName'
   EXEC 'InterfaceScript'
    [OPTS 'OptionsString ...']
```

Policy rule syntax definitions

The GPFS policy rules follow these syntax definitions:

DIRECTORIES_PLUS
: Indicates that non-regular file objects (directories, symbolic links, and so on) should be included in the list. If not specified, only ordinary data files are included in the candidate lists.

DELETE
: Identifies a file deletion rule. A file that matches this rule becomes a candidate for deletion.

EXCLUDE
: Identifies a file exclusion rule. A file that matches this rule is excluded from further rule evaluation. When specified in a LIST rule, indicates that any matching files be excluded from the list.

EXEC *'InterfaceScript'*
: Specifies an external program to be invoked to pass requests to an external storage management application. *InterfaceScript* should be a fully-qualified pathname to a user-provided script or program that supports the commands described in “User provided program for managing external pools” on page 38.

EXTERNAL LIST *ListName*
: Defines an external list. This rule does not match files. It provides the binding between the lists generated with regular **LIST** rules with a matching *ListName* and the external program that you want to run with these lists as input.

EXTERNAL POOL *PoolName*
: Defines an external storage pool. This rule does not match files but serves to define the binding between the policy language and the external storage manager that implements the external storage.

FOR FILESET (*FilesetName***[,***FilesetName***]...)**
: Specifies that the rule should apply only to files within the specified filesets.

FROM POOL *FromPoolName*
Specifies the name of the source pool from which files are candidates for migration.

LIMIT(*OccupancyPercentage***)**
Used to limit the creation of data in a storage pool. If it is determined that transferring the file to the specified pool would exceed the specified occupancy percentage for that pool, GPFS skips the rule and the policy engine looks for the next rule that seems to match. See “Phase two - choosing and scheduling files” on page 32.

For testing or planning purposes, and when using the **mmapplypolicy** command with the **-I defer** or **-I test** options, it is acceptable to specify a **LIMIT** larger than 100%.

LIST *ListName*
Identifies a file list generation rule. A given file may match more than one list rule but will be included in a given list only once. *ListName* provides the binding to an **EXTERNAL LIST** rule that specifies the executable program to use when processing the generated list.

MIGRATE
Identifies a file migration rule. A file that matches this rule becomes a candidate for migration to the pool specified by the **TO POOL** clause.

OPTS ’*OptionsString* **...**’
Specifies optional parameters to be passed to the external program defined with the **EXEC** clause. *OptionsString* is not interpreted by the GPFS policy engine.

REPLICATE (*DataReplication***)**
Overrides the default data replication factor. This value should be specified as 1 or 2.

RESTORE TO POOL *PoolName*
Identifies a file restore rule. When a file is restored using the **gpfs_fputattrswithpathname()** subroutine, this rule allows you to match files against their saved attributes rather than the current file attributes.

RULE [’*RuleName*’**]**
Initiates the rule statement. *RuleName* identifies the rule and is used in diagnostic messages.

SET POOL *PoolName*
Identifies an initial file placement rule. *PoolName* specifies the name of the storage pool where all files that match the rule criteria will be placed.

SHOW ([’*String*’**]** *SqlExpression***)**
Inserts the requested information (the character representation of the evaluated SQL expression *SqlExpression*) into the candidate list created by the rule when it deals with external storage pools. *String* is a literal value that gets echoed back.

This clause has no effect in matching files but can be used to define additional attributes to be exported with the candidate file lists.

THRESHOLD (*HighPercentage***[,***LowPercentage***[,***PremigratePercentage***]])**
Used with the **FROM POOL** clause to control migration and deletion based on the pool storage utilization (percent of assigned storage occupied).

HighPercentage
Indicates that the rule is to be applied only if the occupancy percentage of the named pool is greater than or equal to the *HighPercentage* value. Specify a nonnegative integer in the range 0 to 100.

LowPercentage
Indicates that **MIGRATE** and **DELETE** rules are to be applied until the occupancy percentage of the named pool is reduced to less than or equal to the *LowPercentage* value. Specify a nonnegative integer in the range 0 to 100. The default is 0%.

PremigratePercentage
Defines an occupancy percentage of a storage pool that is below the lower limit. Files that lie between the lower limit *LowPercentage* and the pre-migrate limit *PremigratePercentage* will be copied and become dual-resident in both the internal GPFS storage pool and the designated external storage pool. This option allows the system to free up space quickly by simply deleting pre-migrated files if the pool becomes full. Specify a nonnegative integer in the range 0 to *LowPercentage*. The default is same value as *LowPercentage*.

For more detail on how **THRESHOLD** can be used to control file migration and deletion, see "Phase one - selecting candidate files" on page 31 and "Pre-migrating files with external storage pools" on page 39.

TO POOL *ToPoolName*
Specifies the name of the storage pool where all files that match the rule criteria will be migrated.

WEIGHT (*WeightExpression***)**
Establishes an order on the matching files. Specifies an SQL expression with a numeric value that can be converted to a double precision floating point number. The expression may refer to any of the file attributes and may include any constants and any of the available SQL operators or built-in functions.

WHEN (*TimeBooleanExpression***)**
Specifies an SQL expression that evaluates to **TRUE** or **FALSE**, depending only on the SQL built-in variable **CURRENT_TIMESTAMP**. If the **WHEN** clause is present and *TimeBooleanExpression* evaluates to **FALSE**, the rule is skipped.

The **mmapplypolicy** command assigns the **CURRENT_TIMESTAMP** once, when it begins processing, using either the actual UTC time and date or the date specified with the **-D** option.

WHERE *SqlExpression*
Specifies an SQL expression which may reference file attributes as SQL variables, functions, and operators. Some attributes are not available to all rules. Compares the file attributes specified in the rule with the attributes of the file being created.

SqlExpression must be an expression that will evaluate to **TRUE** or **FALSE**, but can be any combination of standard SQL syntax expressions, including built-in functions.

You can omit the **WHERE** clause entirely. This is equivalent to writing **WHERE TRUE**. When used, the **WHERE** clause must be the last clause of the rule.

SQL expressions for policy rules

A number of the available clauses in the GPFS policy rules utilize SQL expressions. You can reference different file attributes as SQL variables and combine them with SQL functions an operators. Depending on the clause, the SQL expression must evaluate to either **TRUE** or **FALSE**, a numeric value, or a character string. Not all file attributes are available to all rules.

Using file attributes:

The following file attributes can be used in SQL expressions specified with the **WHERE**, **WEIGHT** and **SHOW** clauses:

ACCESS_TIME
Specifies an SQL timestamp value for the date and time that the file was last accessed (POSIX atime).

CHANGE_TIME
Specifies an SQL timestamp value for the date and time that the file metadata was last changed (POSIX ctime).

FILE_SIZE
Specifies the current size or length of the file, in bytes.

FILESET_NAME
Specifies the fileset where the path name for the files is located, or is to be created.

Note: Using the **FOR FILESET** clause has the same effect and is more efficient to evaluate.

GROUP_ID
Specifies the numeric group ID of the file's group.

KB_ALLOCATED
Specifies the number of kilobytes of disk space allocated for the file data.

MISC_ATTRIBUTES
Specifies a variety of miscellaneous file attributes. The value is a concatenated string of attributes that are defined as:

- F - regular data file
- D - directory (to match all directories, you can use 08/08/19 as a wildcard)
- L - symbolic link
- O - other (not F, D, nor L) for example, maybe a device or named pipe
- M - co-managed
- 2 - data blocks are replicated
- I - some data blocks may be ill-placed
- J - some data blocks may by ill-replicated

MODIFICATION_SNAPID
Specifies the integer id of the snapshot after which the file was last changed. The value is normally derived with the **SNAPID()** built-in function which assigns integer values to GPFS snapshot names. This attribute allows policy rules to select files that have been modified since a given snapshot image was taken.

MODIFICATION_TIME
Specifies an SQL timestamp value for the date and time that the file data was last modified (POSIX mtime).

NAME Specifies the name of a file. When used in SQL LIKE comparisons, two wildcard characters are recognized:

- A percent sign (**%**) in the name represents zero or more characters.
- An underscore (_) in the name to represents one single-byte or multibyte character.

PATH_NAME
Specifies a path for the file.

POOL_NAME
Specifies the storage pool where the file data resides.

Note: Using the **FROM POOL** clause has the same effect and is generally preferable.

USER_ID
Specifies the numeric user ID of the owner of the file.

When file attributes are referenced in initial placement rules, only a subset of the attributes is valid, namely: **NAME**, **USER_ID**, **GROUP_ID**, and **FILESET_NAME**. The placement rules, like all rules with a WHERE clause, may also reference the current date and current time and use them to control matching.

When file attributes are used for restoring files, the attributes used for a file correspond to the file's attributes at the time of its backup, not to the current restored file.

Using built-in functions:

With GPFS, you can use built-in functions in comparison predicates, between predicates, in predicates, like predicates, mathematical value expressions, and boolean, string and numeric literals. These functions are organized into three categories:

- "String functions"
- "Numerical functions"
- "Date and time functions" on page 30

String functions: You can use these string-manipulation functions on file names and literals:

Important tip:

1. You must enclose strings in single-quotation marks.
2. You can include a single-quotation mark in a string by using two single-quotation marks. For example, **'a''b'** represents the string **a'b**.

CHAR(x)
Converts an integer x into a string.

CONCAT(x,y)
Concatenates strings x and y.

HEX(x)
Converts an integer x into hexadecimal format.

LENGTH(x)
Determines the length of the data type of string x.

LOWER(x)
Converts string x into lowercase.

SNAPID(SnapshotName)
Convert a snapshot name to an integer to allow numeric comparisons.

Snapshots are identified by the name that was defined when the snapshot was created. Logically, snapshots form an ordered sequence over the changes to a file. In other words, the snapshot that was taken on Monday occurred before the snapshot taken on Tuesday. The policy language converts snapshot names to unique integer IDs to allow numeric comparisons. A more recent snapshot has a higher number than an older snapshot. This allows the rules to determine the files that have changed since a given snapshot was taken and could be used by an incremental backup program to determine the files that have changed since the last backup. For example:

```
WHERE MODIFICATION_SNAPID > SNAPID(YesterdaysSnapshotForBackup)
```

SUBSTR(x,y,z)
Extracts a portion of string x, starting at position y, optionally for z characters (otherwise to the end of the string). This is the short form of **SUBSTRING**.

SUBSTRING(x FROM y FOR z)
Extracts a portion of string x, starting at position y, optionally for z characters (otherwise to the end of the string).

UPPER(x)
Converts the string x into uppercase.

Numerical functions: You can use these numeric-calculation functions to place files based on either numeric parts of the file name, numeric parts of the current date, UNIX-client user IDs or group IDs. These can be used in combination with comparison predicates and mathematical infix operators (such as addition, subtraction, multiplication, division, modulo division, and exponentiation).

INT(x) Converts number x to a whole number, rounding up fractions of .5 or greater.

INTEGER(x)
Converts number x to a whole number, rounding up fractions of .5 or greater.

MOD(x,y)
Determines the value of x taken modulo y (x % y).

Date and time functions: You can use these date-manipulation and time-manipulation functions to place files based on when the files are created and the local time of the GPFS node serving the directory where the file is being created.

CURRENT_DATE
Determines the current date on the GPFS server.

CURRENT_TIMESTAMP
Determines the current date and time on the GPFS server.

DAYOFWEEK(x)
Determines the day of the week from date or timestamp x. The day of a week is from 1 to 7 (Sunday is 1).

DAYOFYEAR(x)
Determines the day of the year from date x. The day of a year is a number from 1 to 366.

DAY(x)
Determines the day of the month from date or timestamp x.

DAYS(x)
Determines the number of days between date or timestamp x and 0001-01-01.

DAYSINMONTH(x)
Determines the number of days in the month of date x.

DAYSINYEAR(x)
Determines the day of the year of date x.

HOUR(x)
Determines the hour of the day (a value from 0 to 23) of timestamp x.

MINUTE(x)
Determines the minute from timestamp x.

MONTH(x)
Determines the month of the year from date or timestamp x.

QUARTER(x)
Determines the quarter of year from date x. Quarter values are the numbers 1 through 4. For example, January, February, and March are in quarter 1.

SECOND(x)
Returns the seconds portion of timestamp x.

WEEK(x)
Determines the week of the year from date x.

YEAR(x)
Determines the year from date or timestamp x.

All date and time functions use Universal Time (UT).

Example

```
/* In this demo GPFS policy rules file some rules are silly */
rule 'stg2' set pool 'TEMP' where lower(NAME) like '%.tmp'
rule 'vip' set pool 'scsi' where USER_ID <= 100
RULE 'm1' SET POOL 'z1' WHERE LOWER(NAME) LIKE '%marc%'
RULE      SET POOL 'z2' REPLICATE (2) WHERE UPPER(NAME) = '%IBM%'
RULE 'r2' SET POOL 'y' WHERE UPPER(SUBSTRING(NAME FROM 1 FOR 4))
                  = 'GPFS'
RULE 'r3' SET POOL 'x' WHERE LOWER(SUBSTR(NAME,1,5))='roger'
```

```
RULE      SET POOL 'z3' WHERE LENGTH(NAME)=7
RULE      SET POOL 'l1' WHERE name like 'xyz%'  AND name like '%qed'
                          OR name like '%.marc.%'
RULE      SET POOL 'l2' WHERE name like 'abc%' OR name like '%xyz'
RULE      SET POOL 'l3' WHERE NOT lower(name) like '%.tmp'
                          AND name like 'x%'
rule 'default' SET POOL 'system'  /* when all else fails ... */
```

Semantics of the mmapplypolicy command and its policy rules

Any given file is a potential candidate for at most one **MIGRATE** or **DELETE** operation during each invocation of the **mmapplypolicy** command. A single invocation of the **mmapplypolicy** command is called the *job*.

The **mmapplypolicy** command sets the SQL built-in variable **CURRENT_TIMESTAMP**, and collects pool occupancy statistics at the beginning of the job.

The **mmapplypolicy** job consists of three major phases:

1. "Phase one - selecting candidate files"
2. "Phase two - choosing and scheduling files" on page 32
3. "Phase three - migrating and deleting files" on page 32

Phase one - selecting candidate files

In the first phase of the **mmapplypolicy** job, all the files within the specified GPFS file system device, or below the input path name, are scanned. The attributes of each file are read from the file's GPFS inode structure. For each file, the policy rules are considered, in order, from first rule to last:

- If the rule has a **WHEN** clause that evaluates to **FALSE**, the rule is skipped.
- If the rule has a **FROM POOL** clause, and the named pool does not match the **POOL NAME** attribute of the file, the rule is skipped.
- If the **FROM POOL** clause is satisfied, but there is also a **THRESHOLD** clause, and if the occupancy percentage of the named pool is less than the *HighPercentage* parameter of the **THRESHOLD** clause, the rule is skipped.
- If the rule has a **FOR FILESET** clause, but none of the named filesets match the **FILESET_NAME** attribute of the file, the rule is skipped.
- If the rule has a **WHERE** clause that evaluates to **FALSE**, the rule is skipped. Otherwise, the rule applies.
- If the applicable rule is an **EXCLUDE** rule, the file will be neither migrated not deleted. Files matching the **EXCLUDE** rule are not candidates for any **MIGRATE** or **DELETE** rule.
- If the applicable rule is a **MIGRATE** rule, the file becomes a *candidate* for migration to the pool specified by the **TO POOL** clause.
- If the applicable rule is a **DELETE** rule, the file becomes a *candidate* for deletion.
- If there is no applicable rule, the file is not a candidate for migration or deletion.
- Each candidate file (for migration or deletion) is also associated with a *LowPercentage* occupancy percentage value, which is taken from the **THRESHOLD** clause of the applicable rule. If not specified, the *LowPercentage* value defaults to 0%.
- Each candidate file is also associated with a numeric *weight*, either computed from the *WeightExpression* of the applicable rule, or assigned a default using these rules:
  - If a *LowPercentage* is specified within a **THRESHOLD** clause of the applicable rule, the weight of the candidate is taken as the **KB_ALLOCATED** attribute of the candidate file.
  - If a *LowPercentage* is not specified within a **THRESHOLD** clause of the applicable rule, the weight of the candidate is taken as **+infinity**.

Phase two - choosing and scheduling files

In the second phase of the **mmapplypolicy** job, some or all of the candidate files are chosen. Chosen files are scheduled for migration or deletion, taking into account the weights and thresholds determined in "Phase one - selecting candidate files" on page 31, as well as the actual pool occupancy percentages. Generally, candidates with higher weights are chosen ahead of those with lower weights.

Generally, a candidate is not chosen for deletion from a pool, nor migration out of a pool, when the pool occupancy percentage falls below the *LowPercentage* value. Also, candidate files will not be chosen for migration into a target **TO POOL** when the target pool reaches the occupancy percentage specified by the **LIMIT** clause (or 99% if no **LIMIT** was explicitly specified by the applicable rule.)

Phase three - migrating and deleting files

In the third phase of the **mmapplypolicy** job, the candidate files that were chosen and scheduled by the second phase are migrated or deleted, each according to its applicable rule. For migrations, if the applicable rule had a **REPLICATE** clause, the replication factors are also adjusted accordingly. It is acceptable for the effective **FROM POOL** and **TO POOL** to be the same because the **mmapplypolicy** command can be used to adjust the replication factors of files without necessarily moving them from one pool to another.

The migration performed in the third phase can involve large amounts of data movement. Therefore, you may want to consider using the **–I defer** option of the **mmapplypolicy** command, and then perform the data movements with the **mmrestripefs -p** command.

Policy rules - examples and tips

Before you write and apply policies, consider this advice:

You are advised to test your rules using the **mmapplypolicy** command with the **-I test** option. Also consider specifying a test-subdirectory within your file system. Do not apply a policy to an entire file system of vital files until you are confident that the rules correctly express your intentions. Even then, you are advised to do a sample run with the **mmapplypolicy -I test** command using the option **-L 3** or higher, to better understand which files are selected as candidates, and which candidates are chosen.

The **-L** flag of the **mmapplypolicy** command can be used to check a policy before it is applied. For examples and more information on this flag, see the section: *The mmapplypolicy -L command* in *General Parallel File System: Problem Determination Guide*.

These examples and tips are provided to illustrate how to perform simple tasks using policy rules with files controlled by GPFS file systems:

1. Each rule begins with the keyword **RULE** and is followed by the name of the rule. Although the rule name is optional, it should be provided and should be unique. Providing a rule name and keeping it unique will help you match error messages with faulty rules.
2. If the storage pool named **pool_1** has an occupancy percentage above 90% now, bring the occupancy percentage of **pool_1** down to 70% by migrating the largest files to storage pool **pool_2**:

   ```
   RULE 'mig1' MIGRATE FROM POOL 'pool_1'
           THRESHOLD(90,70) WEIGHT(KB_ALLOCATED) TO POOL 'pool_2'
   ```

3. Delete files from the storage pool named **pool_1** that have not been accessed in the last 30 days, and are named like temporary files or appear in any directory that is named **tmp**:

   ```
   RULE 'del1' DELETE FROM POOL 'pool_1'
      WHERE (DAYS(CURRENT_TIMESTAMP) - DAYS(ACCESS_TIME) > 30)
           AND (lower(NAME) LIKE '%.tmp' OR PATH_NAME LIKE '%/tmp/%')
   ```

4. Use the SQL **LIKE** predicate to test file names and path names:

   ```
   RULE '*/_*'  DELETE WHERE PATH_NAME LIKE '%/x_%' ESCAPE 'x'
   RULE '*XYZ*' DELETE WHERE NAME LIKE '%XYZ%'
   RULE '12_45' DELETE WHERE NAME LIKE '12x_45' ESCAPE 'x'
   RULE '12%45' DELETE WHERE NAME LIKE '12x%45' ESCAPE 'x'
   ```

```
RULE '12?45' DELETE WHERE NAME LIKE '12_45'
RULE '12*45' DELETE WHERE NAME LIKE '12%45'
RULE '*_*'   DELETE WHERE NAME LIKE '%x_%' ESCAPE 'x'
```

Where:

- A percent **%** wildcard in the name represents zero or more characters.
- An underscore _ wildcard in the name represents one single-byte or multibyte character.

Use the optional **ESCAPE** clause to establish an escape character, when you need to match '_' or '%' exactly.

5. Use the SQL **UPPER** and **LOWER** functions to ignore case when testing names:

```
RULE 'UPPER'  DELETE WHERE upper(PATH_NAME) LIKE '%/TMP/OLD/%'
RULE 'lower'  DELETE WHERE lower(PATH_NAME) LIKE '%/tmp/old/%'
```

6. Use the SQL **SUBSTR** or **SUBSTRING** functions to test a substring of a name:

```
RULE 's1' DELETE WHERE SUBSTRING(NAME FROM 1 FOR 5)='XXXX-'
RULE 's2' DELETE WHERE SUBSTR(NAME,1,5)='YYYY-'
```

7. Use the SQL **SUBSTR** and **LENGTH** functions to test the suffix of a name:

```
RULE 'sfx' DELETE WHERE SUBSTR(NAME,LENGTH(NAME)-3)='.tmp'
```

8. Use a **WHEN** clause to restrict rule applicability to a particular day of the week:

```
RULE 'D_SUN' WHEN (DayOfWeek(CURRENT_DATE)=1) /* Sunday */
  DELETE WHERE PATH_NAME LIKE '%/tmp/%'
```

CURRENT_DATE is an SQL built in operand that returns the date portion of the **CURRENT_TIMESTAMP** value.

9. Use the SQL **IN** operator to test several possibilities:

```
RULE 'D_WEEKEND' WHEN (DayOfWeek(CURRENT_DATE) IN (7,1)) /*  Saturday or Sunday */
  DELETE WHERE PATH_NAME LIKE '%/tmp/%'
```

For information on how to use a macro processor such as **m4** to make reading and writing policy rules easier, see "Using macro processing utilities to simplify policy creation, comprehension, and maintenance" on page 34.

10. Use a **FILESET** clause to restrict the rule to files within particular filesets:

```
RULE 'fsrule1' MIGRATE TO POOL 'pool_2'
         FOR FILESET('root','fset1')
```

In this example there is no **FROM POOL** clause, so regardless of their current storage pool placement, all files from the named filesets are subject to migration to storage pool **pool_2**.

11. Use an **EXCLUDE** rule to exclude a set of files from all subsequent rules:

```
RULE 'Xsuper' EXCLUDE WHERE USER_ID=0
RULE 'mpg'    DELETE  WHERE lower(NAME) LIKE '%.mpg' AND FILE_SIZE>20123456
```

12. Use the SQL **NOT** operator with keywords, along with **AND** and **OR**:

```
RULE 'D_WEEKDAY' WHEN (DayOfWeek(CURRENT_DATE) NOT IN (7,1)) /*  a weekday */
  DELETE WHERE (PATH_NAME LIKE '%/tmp/%' OR NAME LIKE '%.tmp')
          AND (KB_ALLOCATED > 9999 AND NOT USER_ID=0)
```

13. Use a **REPLICATE** clause to increase the availability of selected files:

```
RULE 'R2' MIGRATE FROM POOL 'ypooly' TO POOL 'ypooly'
  REPLICATE(2) WHERE USER_ID=0
```

Before increasing the data replication factor for any file, the file system must be configured to support data replication.

14. By carefully assigning both weights and thresholds, the administrator can formally express rules like this - If the storage pool named **pool_X** has an occupancy percentage above 90% now, bring the occupancy percentage of storage pool named **pool_X** down to 80% by migrating files that are three months or older to the storage pool named **pool_ZZ**. But, if you can find enough year old files to bring the occupancy percentage down to 50%, do that also.

```
RULE 'year-old' MIGRATE FROM POOL 'pool_X'
   THRESHOLD(90,50) WEIGHT(weight_expression)
   TO POOL 'pool_ZZ'
```

```
   WHERE  DAYS(CURRENT_TIMESTAMP) - DAYS(ACCESS_TIME) > 365

RULE '3month-old' MIGRATE FROM POOL 'pool_X'
   THRESHOLD(90,80) WEIGHT(weight_expression)
   TO POOL 'pool_ZZ'
   WHERE  DAYS(CURRENT_TIMESTAMP) - DAYS(ACCESS_TIME) > 90
```

More information about weights is available in the next example.

A goal of this **mmapplypolicy** job is to reduce the occupancy percentage of the **FROM POOL** to the low occupancy percentage specified on the **THRESHOLD** clause, if possible. The **mmapplypolicy** job does not migrate or delete more files than are necessary to produce this occupancy percentage. The task consists of these steps:

a. Each candidate file is assigned a weight.
b. All candidate files are sorted by weight.
c. The highest weight files are chosen to **MIGRATE** or **DELETE** until the low occupancy percentage is achieved, or there are no more candidates.

The administrator who writes the rules must ensure that the computed weights are as intended, and that the comparisons are meaningful. This is similar to the TSM convention, where the weighting function for each file is determined by the equation:

```
X * access_age + Y * file_size
```

where:

access_age is **DAYS(CURRENT_TIMESTAMP) - DAYS(ACCESS_TIME)**

file_size is **FILE_SIZE** or **KB_ALLOCATED**

X and Y are weight factors chosen by the system administrator.

15. The **WEIGHT** clause can be used to express ideas like this (stated informally):

```
IF access_age > 365 days THEN weight = 100000 + access_age
ELSE IF access_age <30 days THEN weight = 0.
ELSE weight= KB_ALLOCATED
```

This means:

- Give a very large weight bias to any file older than a year.
- Force the weight of any file younger than 30 days, to 0.
- Assign weights to all other files according to the number of kilobytes occupied.

The formal SQL syntax is this:

```
CASE
 WHEN DAYS(CURRENT_TIMESTAMP) - DAYS(ACCESS_TIME) > 365
  THEN 100000 + DAYS(CURRENT_TIMESTAMP) - DAYS(ACCESS_TIME)
 WHEN DAYS(CURRENT_TIMESTAMP) - DAYS(ACCESS_TIME) < 30
  THEN 0
 ELSE
  KB_ALLOCATED
END
```

16. The **SHOW** clause has no effect in matching files but can be used to define additional attributes to be exported with the candidate file lists. It may be used for any purpose but is primarily used to support file aggregation.

To support aggregation, you can use the **SHOW** clause to output an aggregation value for each file selected by a rule. You can then output those values to a file list and input that list to an external program that groups the files into aggregates.

Using macro processing utilities to simplify policy creation, comprehension, and maintenance

Prior to evaluating the policy rules, GPFS invokes the **m4** macro processor to process the policy file. This allows you to incorporate into the policy file some of the traditional **m4** facilities and to define simple and

parameterized macros, conditionally include text, perform conditional evaluation, perform simple string operations, perform simple integer arithmetic and much more.

Note: GPFS uses the **m4** built-in **changequote** macro to change the quote pair to [ ] and the **changecom** macro to change the comment pair to /* */ (as in the C programming language).

Utilizing **m4** as a front-end processor simplifies writing policies and produces policies that are easier to understand and maintain. Here is Example 15 on page 34 from “Policy rules - examples and tips” on page 32 written with a few **m4** style macro definitions:

```
define(access_age,(DAYS(CURRENT_TIMESTAMP) - DAYS(ACCESS_TIME)))

define(weight_expression,
       CASE
        WHEN access_age > 365
          THEN 100000 + access_age
        WHEN access_age < 30
          THEN 0
        ELSE
          KB_ALLOCATED
        END
)

RULE year-old MIGRATE FROM POOL pool_X
   THRESHOLD(90,50) WEIGHT(weight_expression)
   TO POOL pool_ZZ
   WHERE access_age > 365

RULE 3month-old MIGRATE FROM POOL pool_X
   THRESHOLD(90,80) WEIGHT(weight_expression)
   TO POOL pool_ZZ
   WHERE access_age > 90
```

If you would like to use megabytes or gigabytes instead of kilobytes to represent file sizes, and **SUNDAY**, **MONDAY**, and so forth instead of 1, 2, and so forth to represent the days of the week, you can use macros and rules like this:

```
define(MB_ALLOCATED,(KB_ALLOCATED/1024.0))
define(GB_ALLOCATED,(KB_ALLOCATED/1048576.0))
define(SATURDAY,7)
define(SUNDAY,1)
define(MONDAY,2)
define(DAY_OF_WEEK, DayOfWeek(CURRENT_DATE))

RULE 'gb1' WHEN(DAY_OF_WEEK IN (SATURDAY,SUNDAY))
        MIGRATE TO POOL 'ypooly' WHERE GB_ALLOCATED >= .015

RULE 'mb4' MIGRATE TO POOL 'zpoolz' WHERE MB_ALLOCATED >= 4
```

The **mmapplypolicy** command provides a **-M** option that can be used to specify **m4** macro definitions when the command is invoked. The policy rules may include variable identifiers whose values can be set using one or more **-M** options on the **mmapplypolicy** command. The policy rules could then compare file attributes to the currently provided values for the macro defined variables.

Among other things, this allows you to create a single policy file and reuse it for incremental backups without editing the file for each backup. For example, if your policy file contains the rules:

```
RULE EXTERNAL POOL 'archive' EXEC '/opts/hpss/archiveScript' OPTS '-server archive_server'
RULE 'mig1' MIGRATE TO POOL 'dead' WHERE ACCESS_TIME < TIMESTAMP(deadline)
RULE 'bak1' MIGRATE TO POOL 'archive' WHERE MODIFICATION_SNAPID > last_snapid
```

Then, if you invoke **mmapplypolicy** with these options

```
mmapplypolicy ... -M "deadline='2006-11-30'" -M "last_snapid=SNAPID('2006_DEC')" \
-M archive_server="archive.abc.com"
```

The "mig1" rule will migrate old files that were not accessed since 2006/11/30 to an online pool named "dead". The "bak1" rule will migrate files that have changed since the 2006_DEC snapshot to an external pool named "archive". When the external script /opts/hpss/archiveScript is invoked, its arguments will include "-server archive.abc.com".

Managing policies

Policies and the rules that they contain are used to assign files to specific storage pools. A storage pool typically contains a set of volumes that provide a specific quality of service for a specific use, such as to store all files for a particular application or a specific business division.

Managing policies includes:

- "Creating a policy"
- "Installing a policy"
- "Changing the active policy"
- "Listing policies" on page 37
- "Validating policies" on page 37
- "Deleting policies" on page 37

Creating a policy

Create a text file for your policy following these guidelines:

- A policy must contain at least one rule.
- A policy file is limited to a size of 1 MB.
- The last placement rule of a policy rule list should be of this form, so that if no other placement rules apply to a file, the file will be assigned to a default pool:

  ```
  RULE 'DEFAULT' SET POOL 'default-data-pool'
  ```

  If you do not do this, and no other rule applies, an **EINVAL** error code is returned.
- Comments within a policy must start with a **/*** and end with a ***/**:

  ```
  /* This is a comment */
  ```

See "Policy rules" on page 24

Installing a policy

To install a policy:

1. Create a text file containing the desired policy rules.
2. Issue the **mmchpolicy** command.

Changing the active policy

Prepare a file with the new or changed policy rules and then issue the **mmchpolicy** command. The **mmchpolicy** command will:

1. Read the policy file into memory and will pass the information to the current file system manager node.
2. The file system manager validates the policy rules.
3. If the policy file contains incorrect rules, no updates are made and an error is returned.
4. If no errors are detected, the new policy rules are installed in an internal file.

Policy changes take effect immediately on all nodes that have the affected file system mounted. For nodes that do not have the file system mounted, policy changes take effect upon the next mount of the file system.

Listing policies

The **mmlspolicy** command displays policy information for a given file system. The information displayed is:

- When the policy file was installed.
- The user who installed the policy file.
- The first line of the original policy file.

The **mmlspolicy -L** command returns the installed (original) policy file. This shows all the rules and comments as they were in the policy file when it was installed. This is useful if you want to change policy rules - simply retrieve the original policy file using the **mmlspolicy -L** command and edit it.

Validating policies

The **mmchpolicy -I test** command validates but does *not* install a policy file.

Deleting policies

To remove the current policy rules and restore the default GPFS file-placement policy, specify **DEFAULT** as the name of the policy file on the **mmchpolicy** command. This is equivalent to installing a policy file with just one rule:

```
RULE 'DEFAULT' SET POOL 'system'
```

Working with external storage pools

Working with external storage pools involves:

- Defining the external pools
- Providing an external program to be used for managing the data and interacting with the external storage management application
- Adding rules that utilize the external pools

Defining external pools

GPFS file management policy rules control data migration into external storage pools. Before you can write a migration policy you must define the external storage pool that the policy will reference. After you define the storage pool, you can then create policies that set thresholds that trigger data migration into or out of the referenced external pool.

When a storage pool reaches the defined threshold or when you invoke **mmapplypolicy**, GPFS processes the metadata, generates a list of files, and invokes a user provided script or program which initiates the appropriate commands for the external data management application to process the files. This allows GPFS to transparently control offline storage and provide a tiered storage solution that includes tape or other media.

Before you can migrate data to an external storage pool, you must define that pool. To define external storage pools, use a GPFS policy rule as follows:

```
RULE EXTERNAL POOL 'PoolName' EXEC 'InterfaceScript' [OPTS  'OptionsString']
```

Where:

- *PoolName* defines the name of the storage pool
- *InterfaceScript* defines the program or script to be invoked to migrate data to or from the external pool
- *OptionsString* is an optional string that, if provided, will be passed to the *InterfaceScript*

You must have a separate EXTERNAL POOL rule for each external pool that you wish to define.

Example:

The following rule defines a storage pool called ″externalpoolA.″

```
RULE EXTERNAL POOL 'externalpoolA' EXEC '/usr/hsm/bin/hsmControl' OPTS '-server=hsm-manager.nyc.com'
```

In this example:

- `externalpoolA` is the name of the external pool
- `/usr/hsm/bin/hsmControl` is the location of the executable script that will be invoked when there are files for migration
- `-server=hsm-manager.nyc.com` is the location of storage pool `externalpoolA`

For additional information, refer to “User provided program for managing external pools.”

User provided program for managing external pools

Once you have defined an external storage pool, subsequent migration or deletion rules may refer to that pool as a source or target storage pool. When the **mmapplypolicy** command is invoked and a rule dictates that data should be moved to or from an external pool, the user provided program identified with the **EXEC** clause in the policy rule launches. That executable program receives three arguments:

- The command to be executed. Your script should implement each of the following sub-commands:
  - LIST - Provides arbitrary lists of files with no semantics on the operation.
  - MIGRATE - Migrate files to external storage and reclaim the online space allocated to the file.
  - PREMIGRATE - Migrate files to external storage but do not reclaim the online space.
  - PURGE - Delete files from both the online file system and the external storage.
  - RECALL - Recall files from external storage to the online storage.
  - TEST – Test for presence and operation readiness. Return zero for success. Return non-zero if the script should not be used on a given node.
- The name of a file containing a list of files to be migrated, pre-migrated, or purged. See “File list format” for detailed description of the layout of the file.
- Any optional parameters specified with the OPTS clause in the rule. These optional parameters are not interpreted by the GPFS policy engine.

The **mmapplypolicy** command invokes the external pool script on all nodes in the cluster that have installed the script in its designated location. The script must be installed at the node that runs **mmapplypolicy**. You can also install the script at other nodes for parallel operation but that is not required. GPFS may call your exit script one or more times for each command.

File list format

Each call to the external pool script specifies the pathname for a temporary file that contains a list of files to be operated on. This file list defines one file per line as follows:

```
InodeNumber GenNumber SnapId [OptionalShowArgs] -- FullPathToFile
```

where:

InodeNumber is a 64-bit inode number.

GenNumber is a 32-bit file generation number.

SnapId is a 64-bit snapshot identifier.

OptionalShowArgs is the result, if any, from the evaluation of the SHOW clause in the policy rule.

FullPathToFile is a fully qualified path name to the file. Files with multiple links can have more than one path, but since the *InodeNumber*, *GenNumber*, and *SnapId* triplet uniquely identifies a GPFS file, only one path will be shown.

The ″--″ characters are a field delimiter that separates the optional show parameters from the path name to the file.

Note: GPFS does not restrict the character set used for path and file names. All characters except '\0' are valid. To make the files readily parsable, files or directories containing the newline character (defined as '\n' in the C programming language) are translated to contain the string ″\n″ instead.

Migrate and recall with external pools

Once you have defined an external storage pool, subsequent migration or deletion rules may refer to that pool as a source or target storage pool. When you invoke **mmapplypolicy** and a rule dictates that data should be deleted or moved to or from an external pool, the program identified in the EXTERNAL POOL rule is invoked with the following arguments:

- The command to be executed.
- The name of the file containing a list of files to be migrated, pre-migrated, or purged.
- Optional parameters, if any.

For example, let us assume an external pool definition:

```
RULE EXTERNAL POOL 'externalpoolA'
   EXEC '/usr/hsm/bin/hsmControl' OPTS '-server=hsm-manager.nyc.com'
```

To move files from the internal **system** pool to storage pool ″externalpoolA″ you would simply define a migration rule that may look something like this:

```
RULE 'MigToExt' MIGRATE FROM POOL('system') TO POOL('externalpoolA') WHERE ...
```

This would result in the external pool script being invoked as follows:

```
/usr/hsm/bin/hsmControl MIGRATE /tmp/filelist -server=hsm-manager.nyc.com
```

Similarly, a rule to migrate data from an external pool back to an internal storage pool could look like:

```
RULE 'MigFromExt' MIGRATE FROM POOL 'externalpoolA' TO POOL 'system' WHERE ...
```

This would result in the external pool script being invoked as follows:

```
/usr/hsm/bin/hsmControl RECALL /tmp/filelist -server=hsm-manager.nyc.com
```

Notes:

1. When migrating to an external storage pool, GPFS ignores the LIMIT and REPLICATION clauses in the policy rule.
2. If you are using HSM with external storage pools, you may need to create specific rules to avoid system problems. These rules should exclude HSM related system files from both migration and deletion. These rules use the form:

   ```
   RULE 'exclude hsm system files' EXCLUDE WHERE PATH_NAME LIKE '%/.SpaceMan%'
   ```

Pre-migrating files with external storage pools

Pre-migration is a standard technique of Hierarchical Storage Management (HSM) systems such as Tivoli Storage Manager. Pre-migration copies data from GPFS internal storage pools to external pools but leaves the original data online in the active file system. Pre-migrated files are often referred to as ″dual resident″ to indicate that the data for the files are available both online in GPFS and offline in the external storage manager. Files in the pre-migrated state allow the external storage manager to respond more quickly to low space conditions by simply deleting the copy of the file data that is stored online.

The files to be pre-migrated are determined by the policy rules that migrate data to an external storage pool. The rule will select files to be migrated and optionally select additional files to be pre-migrated. The THRESHOLD clause of the rule determines the files that need to be pre-migrated.

If you specify the THRESHOLD clause in file migration rules, the **mmapplypolicy** command selects files for migration when the affected storage pool reaches the specified high occupancy percentage threshold. Files are migrated until the storage pool utilization is reduced to the specified low occupancy percentage threshold. When migrating to an external storage pool, GPFS allows you to specify a third pool occupancy

percentage which defines the file pre-migration threshold: after the low occupancy percentage is reached, files are pre-migrated until the pre-migration occupancy percentage is reached.

To explain thresholds in another way, think of an internal storage pool with a high threshold of 90%, a low threshold of 80%, and a pre-migrate threshold of 60%. When this internal storage pool reaches 90% occupancy, the policy rule will migrate files until the occupancy of the pool reaches 80% then it will continue to pre-migrate another 20% of the file space until the 60% threshold is reached.

Pre-migration can only be done with external storage managers using the XDSM Data Storage Management API (DMAPI). Files in the migrated and pre-migrated state will have a DMAPI managed region set on the file data. Files with a managed region are visible to **mmapplypolicy** and may be referenced by a policy rule. You can approximate the amount of pre-migrated space required by counting the space used after the end of the first full data block on all files with managed regions.

Note:

1. If you do not set a pre-migrate threshold or if you set a value that is greater than or equal to the low threshold, then GPFS will not pre-migrate files. This is the default setting.
2. If you set the pre-migrate threshold to zero, then GPFS will pre-migrate all files.

Purging files from external storage pools

Files that have been migrated to an external storage pool continue to have their file name and attributes stored in GPFS; only the file data has been migrated. Files that have been migrated or pre-migrated to an external storage pool may be deleted from the GPFS internal storage pool and from the external storage pool with the policy language using a delete rule.

```
RULE 'DelFromExt' DELETE WHERE ...
```

If the file has been migrated or pre-migrated, this would result in the external pool script being invoked as follows:

```
/usr/hsm/bin/hsmControl PURGE /tmp/filelist -server=hsm-manager.nyc.com
```

The script should delete a file from both the online file system and the external storage manager. However, most HSM systems automatically delete a file from the external storage manager whenever the online file is deleted. If that is how your HSM system functions, your script will only have to delete the online file.

LOW_SPACE and NO_SPACE events

Exhausting space in any one online storage pool generates a NO_SPACE event even though there may be space available in other online storage pools. When the file system runs out of space, DMAPI sends a NO_SPACE event to the storage manager. The storage manager is then expected to migrate data offline to create free space in the online file system. To create free space, file data may be moved to other online storage pools, deleted, or moved to external storage pools. To control these options, the NO_SPACE event is handled by a GPFS daemon process that triggers **mmapplypolicy** to:

- Perform online migration or deletion
- Generate a list of files for migration or deletion
- Invoke the external storage manager if needed

Under most conditions, HSM systems try to avoid NO_SPACE events by monitoring file system space utilization and migrating data offline when the system exceeds a specified threshold. When GPFS reaches a similar threshold, it generates a LOW_SPACE event which also triggers **mmapplypolicy** to generate a list of files to be migrated to external storage.

GPFS uses DMAPI as the signal mechanism for both NO_SPACE and LOW_SPACE events. Therefore, these events are only available if you enable DMAPI in your file system. With DMAPI enabled file systems, a GPFS daemon process, **tsmigrated**, registers for both NO_SPACE and USER events (DMAPI USER events are used to signal LOW_SPACE). Upon receiving an event, GPFS checks for the presence of a

user provided executable script named `/var/mmfs/etc/startpolicy`. If found, GPFS invokes the user script with two parameters: the name of the event (either LOW_SPACE or NO_SPACE) and the name of the file system that generated the event. For example:

```
/var/mmfsetc/startpolicy NO_SPACE fs1
```

If you do not provide an override script, GPFS will invoke the **mmapplypolicy** command to handle the event.

The file with the policy rules used by **mmapplypolicy** is the one that is currently installed in the file system. The HSM user should define migration or deletion rules to reduce the utilization in each online storage pool. Migration rules defined with a high and low THRESHOLD establish the threshold used to signal the LOW_SPACE event for that pool. Since more than one migration rule may be defined, the threshold for a pool is the minimum of the high thresholds set by the rules for that pool. Each pool has its own threshold. Pools without migration rules will not signal a LOW_SPACE event.

Working with external lists

External lists, like external pools, generate lists of files. For external pools, the operations on the files correspond to the rule that references the external pool. For external lists, there is no implied operation; it is simply a list of files that match the criteria specified in the policy rule.

External lists must be defined before they can be used. External lists are defined by:

```
RULE EXTERNAL LIST 'ListName' EXEC 'InterfaceScript' [OPTS 'OptionsString']
```

Where:

- *ListName* defines the name of the external list
- *InterfaceScript* defines the program to be invoked to operate on the list of files
- *OptionsString* is an optional string that, if provided, will be passed to the *InterfaceScript*

Example

The following rule defines an external list called ″listfiles″

```
RULE EXTERNAL LIST 'listfiles' EXEC '/var/mmfs/etc/listControl' OPTS '-verbose'
```

In this example:

- `listfiles` is the name of the external list
- `/var/mmfs/etc/listControl` is the location of the executable script that defines the operations on the list of files
- `-verbose` is an optional flag to the `listControl` script

The EXTERNAL LIST rule provides the binding between the lists generated with regular LIST rules and the external program that you want to run with these lists as input. For example, this rule would generate a list of all files that have more than 1 MB of data in an internal storage pool:

```
RULE 'ListLargeFiles' LIST 'listfiles' WHERE KB_ALLOCATED > 1024
```

By default, only user files are included in lists. To include directories, symbolic links, and other file system objects, the DIRECTORES_PLUS clause must be specified. For example, this rule would generate a list of all objects in the file system.

```
RULE 'ListAllObjects' LIST 'listfiles' DIRECTORIES_PLUS
```

Backup and restore with storage pools

You can use the GPFS ILM tools to backup data for disaster recovery or data archival to an external storage manager such as Tivoli Storage Manager's Backup Archive Client. When backing up data, the external storage manager must preserve the file name, attributes, extended attributes, and the file data.

Among other things, the extended attributes of the file also contain information about the assigned storage pool for the file. When you restore the file, this information is used to assign the storage pool for the file data.

The file data may be restored to the storage pool to which it was assigned when it was backed up or it may be restored to a pool selected by a restore or placement rule using the backed up attributes for the file. GPFS supplies three subroutines that support backup and restore functions with external pools:

- **gpfs_fgetattrs()**
- **gpfs_fputattrs()**
- **gpfs_fputattrswithpathname()**

GPFS exports the extended attributes for a file, including its ACLs, using **gpfs_fgetattrs()**. Included in the extended attributes is the name of the storage pool to which the file has been assigned, as well as file attributes that are used for file placement. When the file is restored the extended attributes are restored using either **gpfs_fputattrs()** or **gpfs_fputattrswithpathname()**.

When you use **gpfs_fputattrs()** to restore the file, GPFS assigns the restored file to the storage pool with the same name as when the file was backed up. Thus by default, restored files are assigned to the same storage pool they were in when they were backed up. If that pool is not available, GPFS tries to select a pool using the currently in effect file placement rules. If that fails, GPFS assigns the file to the system storage pool.

When you restore the file using **gpfs_fputattrswithpathname()**, GPFS is able to access additional file attributes that may have been used by placement or migration policy rules to select the storage pool for the file. This information includes the UID and GID for the owner, the access time for the file, file modification time, file size, the amount of storage allocated, and the full path to the file. GPFS uses **gpfs_fputattrswithpathname()** to match this information with restore policy rules you define.

In other words, the **RESTORE** rule looks at saved file attributes rather than the current file attributes. The call to **gpfs_fputattrswithpathname()** tries to match the saved information to a **RESTORE** rule. If the **RESTORE** rules cannot match saved attributes, GPFS tries to restore the file to the same storage pool it was in when the file was backed up. If that pool is not available GPFS tries to select a pool by matching placement rules. If that fails, GPFS assigns the file to the system storage pool.

Note: When **gpfs_fputattrswithpathname()** is used in conjunction with a **RESTORE** rule and restoring the file to the specified pool would exceed the occupancy percentage defined for that pool, then GPFS skips that rule and the policy engine looks for the next rule that matches. While testing for matching rules, GPFS takes into account the specified replication factor and the **KB_ALLOCATED** attribute of the file being restored.

The **gpfs_fgetattrs()**, **gpfs_fputattrs()** and **gpfs_fputattrswithpathname()** subroutines have several optional flags that you can use to further control the storage pools selection. For detailed information, refer to the *GPFS: Administration and Programming Reference*.

Chapter 3. Creating and maintaining snapshots of GPFS file systems

A snapshot of an entire GPFS file system can be created to preserve the contents of the file system at a single point in time. The storage overhead for maintaining a snapshot is keeping a copy of data blocks that would otherwise be changed or deleted after the time of the snapshot.

Snapshots of a file system are read-only; changes can only be made to the active (that is, normal, non-snapshot) files and directories.

The snapshot function allows a backup or mirror program to run concurrently with user updates and still obtain a consistent copy of the file system as of the time that the snapshot was created. Snapshots also provide an online backup capability that allows easy recovery from common problems such as accidental deletion of a file, and comparison with older versions of a file.

Notes:

1. Because snapshots are not copies of the entire file system, they should not be used as protection against media failures. For information about protection against media failures, see *Recoverability considerations* in the *GPFS: Concepts, Planning, and Installation Guide*.
2. A snapshot of a file creates a new file that captures the user data and user attributes from the original. The snapshot file is independent from the original file. For DMAPI managed file systems, the snapshot of a file is not automatically managed by DMAPI, regardless of the state of the original file. The DMAPI attributes from the original file are not inherited by the snapshot. For more information about DMAPI restrictions for GPFS, see the *GPFS: Data Management API Guide*.

Management of snapshots of a GPFS file system include:

Creating your GPFS snapshot

Use the **mmcrsnapshot** command to create a snapshot of an entire GPFS file system at a single point in time. Snapshots appear in the file system tree as hidden subdirectories of the root.

If you prefer to create and delete invisible directories that connect to the snapshots of a GPFS file system, you can use the **mmsnapdir** command. See "Linking to your GPFS snapshots" on page 46.

Each snapshot of the file system, *Device*, is identified by *Directory* name on the **mmcrsnapshot** command. For example, given the file system **fs1** to create a snapshot **snap1**, enter:

```
mmcrsnapshot fs1 snap1
```

The output is similar to this:

```
Writing dirty data to disk
Quiescing all file system operations
Writing dirty data to disk again
Creating snapshot.
Resuming operations.
```

Before issuing the command, the directory structure would appear similar to:

```
/fs1/file1
/fs1/userA/file2
/fs1/userA/file3
```

After the command has been issued, the directory structure would appear similar to:

```
/fs1/file1
/fs1/userA/file2
/fs1/userA/file3

/fs1/.snapshots/snap1/file1
/fs1/.snapshots/snap1/userA/file2
/fs1/.snapshots/snap1/userA/file3
```

If a second snapshot were to be created at a later time, the first snapshot would remain as is. Snapshots are only made of active file systems, not existing snapshots. For example:

```
mmcrsnapshot fs1 snap2
```

The output is similar to this:

```
Writing dirty data to disk
Quiescing all file system operations
Writing dirty data to disk again
Creating snapshot.
Resuming operations.
```

After the command has been issued, the directory structure would appear similar to:

```
/fs1/file1
/fs1/userA/file2
/fs1/userA/file3

/fs1/.snapshots/snap1/file1
/fs1/.snapshots/snap1/userA/file2
/fs1/.snapshots/snap1/userA/file3

/fs1/.snapshots/snap2/file1
/fs1/.snapshots/snap2/userA/file2
/fs1/.snapshots/snap2/userA/file3
```

See the **mmcrsnapshot** command for complete usage information.

Listing GPFS snapshots

Use the **mmlssnapshot** command to display existing snapshots of a file system and their attributes.

Use these attributes to:

-d Display the amount of storage used by a snapshot

GPFS quota management does not take the data blocks used to store snapshots into account when reporting on and determining if quota limits have been exceeded.

-Q Display whether or not quotas were automatically activated at mount time when the snapshot was created.

When a snapshot is restored with the **mmrestorefs** command, it may be necessary to restore the quota activation with the **mmchfs -Q** command.

For example, to list the snapshots for file system **fs1**, enter:

```
mmlssnapshot fs1 -L
```

The system displays information similar to:

```
Snapshots in file system fs1: [data and metadata in KB]
Directory               SnapId    Status          Created                        Data  Metadata
snap1                   1         Valid           Fri Oct 17 10:56:22 2003          0       512
```

See the **mmlssnapshot** command for complete usage information.

Restoring a GPFS file system from a snapshot

Use the **mmrestorefs** command to restore user data and attribute files in an active file system from the specified snapshot.

Prior to issuing the **mmrestorefs** command, you must unmount the file system from all nodes in the cluster. The file system may not be remounted until the **mmrestorefs** command has successfully completed, unless you have specified the **-c** option to force the restore to continue even in the event errors are encountered.

Existing snapshots, including the one being used in the restore, are not affected by the **mmrestorefs** command. To obtain a snapshot of the restored file system, you must reissue the **mmcrsnapshot** command.

As an example, let us assume we have a directory structure similar to:

```
/fs1/file1
/fs1/userA/file2
/fs1/userA/file3
/fs1/.snapshots/snap1/file1
/fs1/.snapshots/snap1/userA/file2
/fs1/.snapshots/snap1/userA/file3
```

If the directory **userA** is then deleted, we would have:

```
/fs1/file1
/fs1/.snapshots/snap1/file1
/fs1/.snapshots/snap1/userA/file2
/fs1/.snapshots/snap1/userA/file3
```

If the directory **userB** is then created using the inode originally assigned to **userA** and we take another snapshot:

```
mmcrsnapshot fs1 snap2
```

The output is similar to this:

```
Writing dirty data to disk
Quiescing all file system operations
Writing dirty data to disk again
Creating snapshot.
Resuming operations.
```

After the command is issued, the directory structure would appear similar to:

```
/fs1/file1
/fs1/userB/file2b
/fs1/userB/file3b
/fs1/.snapshots/snap1/file1
/fs1/.snapshots/snap1/userA/file2
/fs1/.snapshots/snap1/userA/file3
/fs1/.snapshots/snap2/file1
/fs1/.snapshots/snap2/userB/file2b
/fs1/.snapshots/snap2/userB/file3b
```

If the file system is then restored from **snap1**:

```
mmrestorefs fs1 snap1
```

After the command is issued, the directory structure would appear similar to:

```
/fs1/file1
/fs1/userA/file2
/fs1/userA/file3
/fs1/.snapshots/snap1/file1
/fs1/.snapshots/snap1/userA/file2
/fs1/.snapshots/snap1/userA/file3
/fs1/.snapshots/snap2/file1
/fs1/.snapshots/snap2/userB/file2b
/fs1/.snapshots/snap2/userB/file3b
```

See the **mmrestorefs** command for complete usage information.

Linking to your GPFS snapshots

Snapshot root directories appear in a special **.snapshots** directory under the file system root. If you prefer to link directly to the snapshot rather than always traverse the root directory, you can use the **mmsnapdir** command to add a **.snapshots** subdirectory to all directories in the file system.

These **.snapshots** subdirectories will contain a link into the corresponding directory of each snapshot that exists for the file system.

Unlike **.snapshots** in the root directory, however, the **.snapshots** directories added by the **mmsnapdir** command are invisible in the sense that the **ls** command or **readdir()** function does not return **.snapshots**. This is to prevent recursive file system utilities such as **find** or **tar** from entering into the snapshot tree for each directory they process. For example, **ls -a /fs1/userA** does not show **.snapshots**, but **ls /fs1/userA/.snapshots** and **cd /fs1/userA/.snapshots** do show **.snapshots**. If a user wants to make one of their snapshot directories more visible, it is suggested to create a symbolic link to **.snapshots**.

Specifying the **-r** option on the **mmsnapdir** command will change back to the default behavior – a single **.snapshots** directory in the file system root directory. The **-s** option allows you to change the name of the **.snapshots** directory. See the **mmsnapdir** command for complete usage information.

To illustrate the above, let us assume a GPFS file system **fs1** which is mounted at **/fs1** and has one snapshot, **snap1**. The file system may appear something like this:

```
/fs1/userA/file2b
/fs1/userA/file3b
/fs1/.snapshots/snap1/userA/file2
/fs1/.snapshots/snap1/userA/file3
```

To create links to the snapshots from each directory, and instead of **.snapshots**, use the name **.links**, enter:

```
mmsnapdir fs1 -a -s .links
```

After the command completes, the directory structure would appear similar to:

```
/fs1/userA/file2b
/fs1/userA/file3b
/fs1/userA/.links/snap1/file2
/fs1/userA/.links/snap1/file3

/fs1/.links/snap1/userA/file2
/fs1/.links/snap1/userA/file3
```

To delete the links, issue:

```
mmsnapdir fs1 -r
```

After the command completes, the directory structure would appear similar to:

```
/fs1/userA/file2b
/fs1/userA/file3b

/fs1/.links/snap1/userA/file2
/fs1/.links/snap1/userA/file3
```

Deleting your GPFS snapshot

Use the **mmdelsnapshot** command to delete GPFS snapshots of a file system.

For example, to delete **snap1** for the file system **fs1**, enter:

```
mmdelsnapshot fs1 snap1
```

The output is similar to this:

```
Deleting snapshot files...
Delete snapshot snap1 complete, err = 0
```

See the **mmdelsnapshot** command in the *GPFS: Administration and Programming Reference* for complete usage information.

Chapter 4. Establishing disaster recovery for your GPFS cluster

The ability to detect and quickly recover from a massive hardware failure is of paramount importance to businesses that make use of real-time data processing systems.

GPFS provides a number of features that facilitate the implementation of highly-available GPFS environments capable of withstanding catastrophic hardware failures. By maintaining a replica of the file system's data at a geographically-separate location, the system sustains its processing using the secondary replica of the file system in the event of a total failure in the primary environment.

On a very high level, a disaster-resilient GPFS cluster is made up of two or three, distinct, geographically-separate hardware sites operating in a coordinated fashion. Two of the sites consist of GPFS nodes and storage resources holding a complete replica of the file system. If a third site is active, it consists of a single node and a single disk used as a tiebreaker for GPFS quorum. In the event of a catastrophic hardware failure that disables the operation of an entire site, and assuming the tiebreaker site remains operational, file system services failover to the remaining subset of the cluster and continue serving the data using the replica of the file system that survived the disaster. However if the tiebreaker fails during the disaster, the remaining number of nodes and disks is insufficient to satisfy the quorum rules and the surviving site loses access to the GPFS file system. A manual procedure is needed to instruct GPFS to disregard the existing quorum assignments and continue operating with whatever resources are available.

The secondary replica is maintained by one of several methods:

- Synchronous mirroring utilizing GPFS replication.

  The data and metadata replication features of GPFS are used to implement synchronous mirroring between a pair of geographically-separate sites. The use of logical replication-based mirroring offers a generic solution that relies on no specific support from the disk subsystem beyond the basic ability to read and write data blocks. See *Synchronous mirroring utilizing GPFS replication*.
- Synchronous mirroring utilizing IBM TotalStorage® Enterprise Storage Server® (ESS) Peer-to-Peer Remote Copy (PPRC).

  The PPRC feature of the ESS establishes a persistent mirroring relationship between pairs of Logical Units (LUNs) on two subsystems connected over an ESCON® or a fiber-channel link. All updates performed on the set of primary, or source, LUNs appear in the same order on the secondary, or target, disks in the target subsystem. The PPRC mechanism provides for an exact bitwise replica of the source's content as seen at the time of the failure on the target should the source volume fail. See *Synchronous mirroring utilizing IBM TotalStorage ESS PPRC*.

  Usage of synchronous IBM TotalStorage Enterprise Storage Server (ESS) Peer-to-Peer Remote Copy (PPRC), now extends to IBM TotalStorage Metro Mirror.

  Usage of asynchronous PPRC, now extends to IBM TotalStorage Global Mirror.
- Asynchronous mirroring utilizing ESS FlashCopy®.

  Periodic point-in-time copies of the file system are taken using the facilities of ESS FlashCopy. The copy is subsequently transferred to a remote backup location using PPRC, written to tape, or both. See *Asynchronous mirroring utilizing ESS FlashCopy*.

The primary advantage of both synchronous mirroring methods is the minimization of the risk of permanent data loss. Both methods provide two consistent, up-to-date replicas of the file system, each available for recovery should the other one fail. However, inherent to all solutions that synchronously mirror data over a wide area network link is the latency penalty induced by the replicated write I/Os. This makes both synchronous mirroring methods prohibitively inefficient for certain types of performance-oriented applications. The asynchronous method effectively eliminates this penalty. However, asynchronous mirroring may result in two distinct and not necessarily consistent replicas of the file system. There are no

guarantees as to the validity of data kept in the snapshot beyond the fact that the file system's metadata is consistent and that the user data must have been valid at the time the snapshot was taken.

Synchronous mirroring utilizing GPFS replication

In a configuration utilizing GPFS replication, a single GPFS cluster is defined over three geographically-separate sites consisting of two production sites and a third tiebreaker site. One or more file systems are created, mounted, and accessed concurrently from the two active production sites.

The data and metadata replication features of GPFS are used to maintain a secondary copy of each file system block, relying on the concept of disk failure groups to control the physical placement of the individual copies:

1. Separate the set of available disk volumes into two failure groups. Define one failure group at each of the active production sites.
2. Create a replicated file system. Specify a replication factor of 2 for both data and metadata.

When allocating new file system blocks, GPFS always assigns replicas of the same block to distinct failure groups. This provides a sufficient level of redundancy allowing each site to continue operating independently should the other site fail.

GPFS enforces a node quorum rule to prevent multiple nodes from assuming the role of the file system manager in the event of a network partition. Thus, a majority of quorum nodes must remain active in order for the cluster to sustain normal file system usage. Furthermore, GPFS uses a quorum replication algorithm to maintain the content of the file system descriptor (one of the central elements of the GPFS metadata). When formatting the file system, GPFS assigns some number of disks (usually three) as the descriptor replica holders that are responsible for maintaining an up-to-date copy of the descriptor. Similar to the node quorum requirement, a majority of the replica holder disks must remain available at all times to sustain normal file system operations. This file system descriptor quorum is internally controlled by the GPFS daemon. However, when a disk has failed due to a disaster you must manually inform GPFS that the disk is no longer available and it should be excluded from use.

Considering these quorum constraints, it is suggested a third site in the configuration fulfills the role of a tiebreaker for the node and the file system descriptor quorum decisions. The tiebreaker site consists of:

1. A single quorum node

   As the function of this node is to serve as a tiebreaker in GPFS quorum decisions, it does not require normal file system access and SAN connectivity. To ignore disk access errors on the tiebreaker node, enable the **unmountOnDiskFail** configuration parameter through the **mmchconfig** command . When enabled, this parameter forces the tiebreaker node to treat the lack of disk connectivity as a local error, resulting in a failure to mount the file system, rather that reporting this condition to the file system manager as a disk failure.
2. A single network shared disk

   The function of this disk is to provide an additional replica of the file system descriptor file needed to sustain quorum should a disaster cripple one of the other descriptor replica disks. Create a network shared disk over the tiebreaker node's internal disk defining:

   - the local node as an NSD server
   - the disk usage as **descOnly**

     The **descOnly** option instructs GPFS to only store file system descriptor information on the disk.

This three-site configuration is resilient to a complete failure of any single hardware site. Should all disk volumes in one of the failure groups become unavailable, GPFS performs a transparent failover to the remaining set of disks and continues serving the data to the surviving subset of nodes with no administrative intervention. While nothing prevents you from placing the tiebreaker resources at one of the active sites, to minimize the risk of double-site failures it is suggested you install the tiebreakers at a third, geographically distinct location.

The high-level organization of a replicated GPFS cluster for synchronous mirroring where all disks are directly attached to all nodes in the cluster is shown in Figure 5. An alternative to this design would be to have the data served through designated NSD servers.

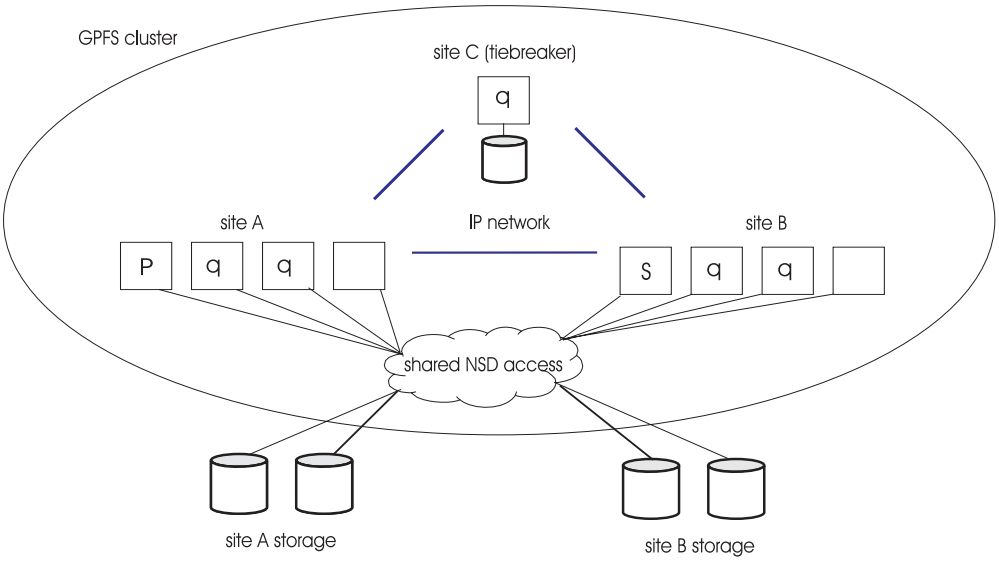


Figure 5. Synchronous mirroring utilizing GPFS replication

Setting up a GPFS cluster with synchronous mirroring utilizing GPFS replication

To establish a disaster-resilient GPFS cluster utilizing replication as shown in Figure 5, consider the configuration:

Site A Consisting of:

- Nodes – **nodeA001**, **nodeA002**, **nodeA003**, **nodeA004**
- Disk device names – **diskA1**, **diskA2**

  diskA1 and **diskA2** are SAN-attached and accessible from all nodes at site **A** and site **B**.

Site B Consisting of:

- Nodes – **nodeB001**, **nodeB002**, **nodeB003**, **nodeB004**
- Disks – **diskB1**, **diskB2**

  diskB1 and **diskB2** are SAN-attached and accessible from all nodes at site **A** and site **B**.

Site C (tiebreaker)

Consisting of:

- Node – **nodeC**
- Disk – **diskC**

  diskC is a NSD defined over the internal disk of the node **nodeC** and is directly accessible only from site **C**

1. Create a GPFS cluster selecting **nodeA001** at site A as the primary cluster data server node, **nodeB001** at site **B** as the secondary cluster data server nodes, and the nodes in the cluster contained in the file **clusterNodes**. The **clusterNodes** file contains the node descriptors:

```
nodeA001:quorum-manager
nodeA002:quorum-manager
nodeA003:quorum-manager
nodeA004:client
nodeB001:quorum-manager
nodeB002:quorum-manager
nodeB003:quorum-manager
nodeB004:client
nodeC:quorum-client
```

Issue this command:

```
mmcrcluster –N clusterNodes –p nodeA001 –s nodeB001
```

2. Prevent false disk errors in the SAN configuration from being reported to the file system manager by enabling the **unmountOnDiskFail** option on the tiebreaker node:

   ```
   mmchconfig unmountOnDiskFail=yes -N nodeC
   ```

3. Define the set of network shared disks for the cluster where disks at sites **A** and **B** are assigned to failure groups 1 and 2, respectively. The tiebreaker disk is assigned to failure group 3. The disk descriptors contained in the file **clusterDisks** are:

   ```
   /dev/diskA1:nodeA002:nodeA003:dataAndMetadata:1
   /dev/diskA2:nodeA002:nodeA003:dataAndMetadata:1
   /dev/diskB1:nodeB002:nodeB003:dataAndMetadata:2
   /dev/diskB2:nodeB002:nodeB003:dataAndMetadata:2
   /dev/diskC1:nodeC::descOnly:3
   ```

   Issue this command:

   ```
   mmcrnsd –F clusterDisks
   ```

4. Issue the **mmlsnsd** command to verify that the network shared disks have been created:

   ```
   mmlsnsd -m
   ```

   Output is similar to this:

   ```
   Disk name    NSD volume ID      Device         Node name         Remarks
   ---------------------------------------------------------------------------
    gpfs1nsd     0972445B416BE502   /dev/diskA1    nodeA002          server node
    gpfs1nsd     0972445B416BE502   /dev/diskA1    nodeA003          server node
    gpfs2nsd     0972445B416BE509   /dev/diskA2    nodeA002          server node
    gpfs2nsd     0972445B416BE509   /dev/diskA2    nodeA003          server node
    gpfs3nsd     0972445F416BE4F8   /dev/diskB1    nodeB002          server node
    gpfs3nsd     0972445F416BE4F8   /dev/diskB1    nodeB003          server node
    gpfs4nsd     0972445F416BE4FE   /dev/diskB2    nodeB002          server node
    gpfs4nsd     0972445F416BE4FE   /dev/diskB2    nodeB003          server node
    gpfs5nsd     0972445D416BE504   /dev/diskC1    nodeC             server node
   ```

5. Start the GPFS daemon on all nodes:

   ```
   mmstartup -a
   ```

6. Create a replicated file system **fs0**:

   ```
   mmcrfs /gpfs/fs0 fs0 –F clusterDisks –m 2 –M 2 –r 2 –R 2
   ```

7. Mount **fs0** on all nodes at sites A and B.

Steps to take after a disaster when using GPFS replication

Utilizing GPFS replication allows for *failover* to the surviving site without disruption of service as long as both the remaining site and the tiebreaker site remain functional. It remains in this state until a decision is made to restore the operation of the affected site by executing the *fail-back* procedure. If the tiebreaker site is also affected by the disaster and is no longer operational, GPFS quorum is broken and manual intervention is required to resume file system access. Existing quorum designations must be relaxed in order to allow the surviving site to fulfill quorum requirements:

1. To relax node quorum, temporarily change the designation of each of the failed quorum nodes to non-quorum nodes. Issue the **mmchnode --nonquorum** command.
2. To relax file system descriptor quorum, temporarily eliminate the failed disks from the group of disks from which the GPFS daemon uses to write the file system descriptor file to. Issue the **mmfsctl exclude** command for each of the failed disks.

While the GPFS cluster is in a failover state it is suggested that no changes to the GPFS configuration are made. Changes to your GPFS configuration require both cluster configuration servers to be operational. If both servers are not operational, the sites would have distinct, and possibly inconsistent, copies of the GPFS **mmsdrfs** configuration data file. While the servers can be migrated to the surviving site, it is best to avoid this step if the disaster does not leave the affected site permanently disabled.

If it becomes absolutely necessary to modify the GPFS configuration while in failover mode, for example to relax quorum, you must ensure that all nodes at the affected site are powered down and left in a stable inactive state. They must remain in such state until the decision is made to execute the fail-back procedure. As a means of precaution, we suggest disabling the GPFS autoload option on all nodes to prevent GPFS from bringing itself up automatically on the affected nodes should they come up spontaneously at some point after a disaster.

Failover to the surviving site

Following a disaster, which failover process is implemented depends upon whether or not the tiebreaker site is affected:

- **Failover without the loss of tiebreaker site C**

  The proposed three-site configuration is resilient to a complete failure of any single hardware site. Should all disk volumes in one of the failure groups become unavailable, GPFS performs a transparent failover to the remaining set of disks and continues serving the data to the surviving subset of nodes with no administrative intervention.
- **Failover with the loss of tiebreaker site C**

  If both site **A** and site **C** fail:

  1. Shut the GPFS daemon down on the surviving nodes at site **B**, where the file **gpfs.siteB** lists all of the nodes at site **B**:

     ```
     mmshutdown -N gpfs.siteB
     ```
  2. If it is necessary to make changes to the configuration, migrate the primary cluster configuration server to a node at site **B**:

     ```
     mmchcluster -p nodeB002
     ```
  3. Relax node quorum by temporarily changing the designation of each of the failed quorum nodes to non-quorum nodes:

     ```
     mmchnode --nonquorum -N nodeA001,nodeA002,nodeA003,nodeC
     ```
  4. Relax file system descriptor quorum by informing the GPFS daemon to migrate the file system descriptor off of the failed disks:

     ```
     mmfsctl fs0 exclude -d "gpfs1nsd;gpfs2nsd;gpfs5nsd"
     ```
  5. Restart the GPFS daemon on the surviving nodes:

     ```
     mmstartup -N gpfs.siteB
     ```
  6. Mount the file system on the surviving nodes at site B.

Fail-back procedures

Which failback procedure you follow depends upon whether the nodes and disks at the affected site have been repaired or replaced. If the disks have been repaired, you must also consider the state of the data on the failed disks:

- For nodes and disks that have been repaired and *you are certain* the data on the failed disks has not been changed, follow either:

- *Fail-back with temporary loss and no configuration changes*
- *Fail-back with temporary loss and configuration changes*

- If the nodes have been replaced and either the disks have been replaced or repaired, and *you are not certain* the data on the fail disks has not been changed, follow the procedure for *Fail-back with permanent loss*.

Delayed failures: In certain failure cases the loss of data may not be immediately apparent. For example, consider this sequence of events:

1. Site **B** loses connectivity with sites **A** and **C**.
2. Site **B** then goes down due to loss of node quorum.
3. Sites **A** and **C** remain operational long enough to modify some of the data on disk but suffer a disastrous failure shortly afterwards.
4. Node and file system descriptor quorums are overridden to enable access at site **B**.

Now the two replicas of the file system are inconsistent and the only way to reconcile these copies during recovery is to:

1. Remove the damaged disks at sites **A** and **C**.
2. Either replace the disk and format a new NSD or simply reformat the existing disk if possible.
3. Add the disk back to the file system, performing a full resynchronization of the file system's data and metadata and restore the replica balance using the **mmrestripefs** command.

Fail-back with temporary loss and no configuration changes:

If the outage was of a temporary nature and your configuration has not been altered, it is a simple process to fail-back to the original state. After all affected nodes and disks have been repaired and *you are certain* the data on the failed disks has not been changed:

1. Start GPFS on the repaired nodes where the file **gpfs.sitesAC** lists all of the nodes at sites **A** and **C**:
   ```
   mmstartup -N gpfs.sitesAC
   ```
2. Restart the affected disks. If more than one disk in the file system is down, they must all be started at the same time:
   ```
   mmchdisk fs0 start -a
   ```

Fail-back with temporary loss and configuration changes:

If the outage was of a temporary nature and your configuration has been altered, follow this procedure to fail-back to the original state. After all affected nodes and disks have been repaired and *you are certain* the data on the failed disks has not been changed:

1. Ensure that all nodes have the latest copy of the **mmsdrfs** file:
   ```
   mmchcluster -p LATEST
   ```
2. Migrate the primary cluster configuration server back to site **A**:
   ```
   mmchcluster -p nodeA001
   ```
3. Restore node quorum designations at sites **A** and **C**:
   ```
   mmchnode --quorum -N nodeA001,nodeA002,nodeA003,nodeC
   ```
4. Start GPFS on the repaired nodes where the file **gpfs.sitesAC** lists all of the nodes at sites **A** and **C**:
   ```
   mmstartup -N gpfs.sitesAC
   ```
5. Restore the file system descriptor quorum by informing the GPFS to include the repaired disks:
   ```
   mmfsctl fs0 include -d "gpfs1nsd;gpfs2nsd;gpfs5nsd"
   ```
6. Bring the disks online and restripe the file system across all disks in the cluster to restore the initial replication properties:

```
mmchdisk fs0 start -a
mmrestripefs fs0 -b
```

the **-r** flag may be used on the **mmrestripefs** command instead.

Fail-back with permanent loss:

If the outage is of a permanent nature:

1. Remove the failed resources from the GPFS configuration
2. Replace the failed resources, then add the new resources into the configuration
3. Resume the operation of GPFS across the entire cluster

Assume that sites **A** and **C** have had permanent losses. To remove all references of the failed nodes and disks from the GPFS configuration and replace them :

1. To remove the failed resources from the GPFS configuration:
   a. If as part of the failover process, *you did not* migrate the primary cluster configuration server, migrate the server to node **nodeB002** at site B:
      ```
      mmchcluster –p nodeB002
      ```
   b. Delete the failed disks from the GPFS configuration:
      ```
      mmdeldisk fs0 "gpfs1nsd;gpfs2nsd;gpfs5nsd"
      mmdelnsd "gpfs1nsd;gpfs2nsd;gpfs5nsd"
      ```
   c. Delete the failed nodes from the GPFS configuration:
      ```
      mmdelnode -N nodeA001,nodeA002,nodeA003,nodeA004,nodeC
      ```
2. If there are new resources to add to the configuration:
   a. Add the new nodes at sites **A** and **C** to the cluster where the file **gpfs.sitesAC** lists of the new nodes:
      ```
      mmaddnode -N gpfs.sitesAC
      ```
   b. Ensure that all nodes have the latest copy of the **mmsdrfs** file:
      ```
      mmchcluster -p LATEST
      ```
   c. Migrate the primary cluster configuration server back to site **A**:
      ```
      mmchcluster -p nodeA001
      ```
   d. Start GPFS on the new nodes
      ```
      mmstartup -N gpfs.sitesAC
      ```
   e. Prepare the new disks for use in the cluster, create the NSDs using the original disk descriptors for site **A** contained in the file **clusterDisksAC**:
      ```
      /dev/diskA1:nodeA002:nodeA003:dataAndMetadata:1
      /dev/diskA2:nodeA002:nodeA003:dataAndMetadata:1
      /dev/diskC1:nodeC::descOnly:3
      ```

      Issue this command:
      ```
      mmcrnsd -F clusterDisksAC
      ```
   f. Add the new NSDs to the file system specifying the **-r** option to rebalance the data on all disks:
      ```
      mmadddisk fs0 -F clusterDisksAC -r
      ```

Synchronous mirroring utilizing IBM TotalStorage ESS PPRC

PPRC is a function that continuously updates a secondary (target) copy of an ESS disk volume to match changes made to a primary (source) volume. Any pair of equal-sized ESS disks can be configured for a PPRC relationship, during which all write operations performed on the source are synchronously mirrored to the target device.

The PPRC protocol guarantees that the secondary copy is constantly up-to-date by ensuring that the primary copy is written only if the primary storage subsystem received acknowledgement that the secondary copy has been written. The paired volumes typically reside on two distinct and geographically separated ESS devices communicating over ESCON or over a fiber channel link.

A number of PPRC tasks are provided to facilitate recovery in the event of a site-wide failure. After the failure of the primary volume (or the failure of the entire storage subsystem), users execute the PPRC failover task, which suspends the PPRC relationship between the given pair of volumes and turns the target volume into a primary. When a volume enters the suspended state, a modification bitmap is established to keep track of the write operations performed on that volume to allow for an efficient resynchronization.

Once the operation of the original primary volume has been restored, the PPRC fail-back task is executed to resynchronize the content of the two volumes. The original source volume is switched to the target mode, after which all modified data tracks (those recorded in the modification bitmap) are copied from the original target disk. The volume pair is then suspended again and another task is executed to reverse the volumes' roles, thus bringing the pair into its initial state.

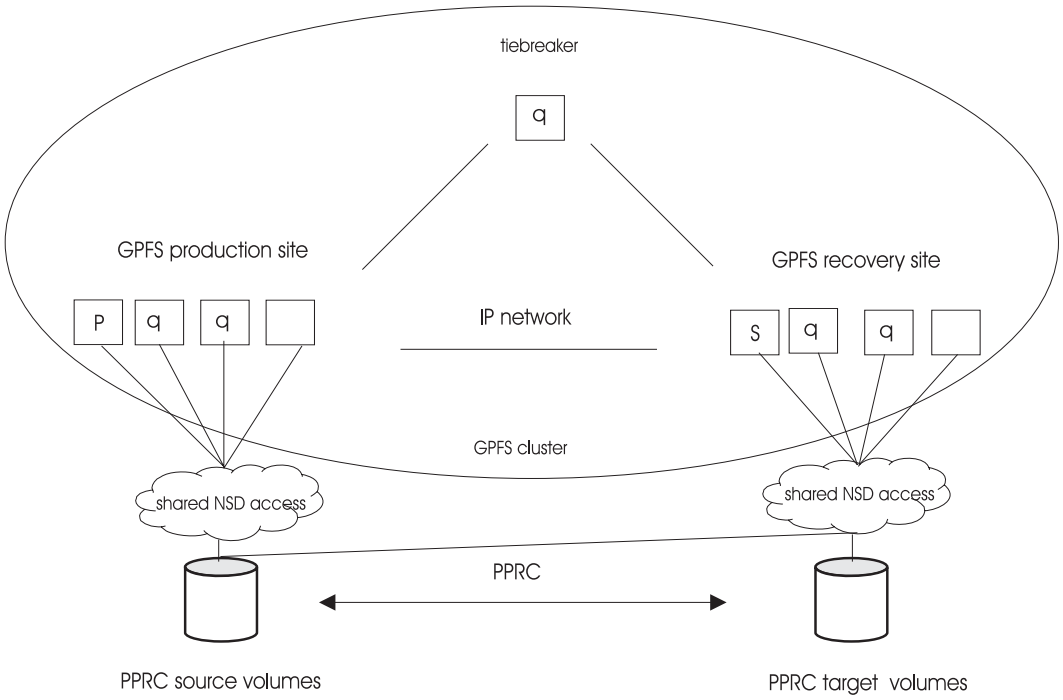
The ESS Copy Services Web Interface User's Guide as described in the *IBM TotalStorage Enterprise Storage Server Web Interface User's Guide*, is a GUI that allows users to establish and terminate PPRC pairs, and invoke failover, fail-back, and related PPRC functions. A Java-based command-line interface as described in the *IBM Enterprise Storage Server Command-Line Interfaces User's Guide*, provides another method of interaction.

A PPRC-based GPFS cluster can be established in two manners:

- A single GPFS cluster encompassing two sites and an optional tiebreaker site
- Two distinct GPFS clusters

An active/active GPFS cluster

The high-level organization of PPRC-based active/active GPFS cluster is illustrated in Figure 6 on page 57. A single GPFS cluster is created over three sites. The data is mirrored between two active sites with a cluster configuration server residing at each site and a tiebreaker quorum node installed at the third location. The presence of an optional tiebreaker node allows the surviving site to satisfy the node quorum requirement with no additional intervention. Without the tiebreaker, the failover procedure requires an additional administrative command to relax node quorum and allow the remaining site to function independently. Furthermore, the nodes at the recovery site have direct disk paths to the primary site's storage.



P - primary cluster configuration server

S - secondary cluster configuration server

q - quorum node

Figure 6. A synchronous active/active PPRC-based mirrored GPFS configuration with a tiebreaker site

Setting up an active/active GPFS configuration

To establish an active/active PPRC-based GPFS cluster with a tiebreaker site as shown in Figure 6, consider the configuration:

Site A (production site)

Consists of:

- Nodes – **nodeA001**, **nodeA002**, **nodeA003**, **nodeA004**
- Storage subsystems – Enterprise Storage Server (ESS) **A**, logical subsystem (LSS) **A**
- Disk volumes – **diskA** on LSS **A**

  diskA is SAN-attached and accessible from sites **A** and **B**

Site B (recovery site)

Consists of:

- Nodes – **nodeB001**, **nodeB002**, **nodeB003**, **nodeB004**
- Storage subsystems – ESS **B**, LSS **B**
- Disk volumes – **diskB** on LSS **B**

  diskB is SAN-attached and accessible from site **B** only

Site C (tiebreaker)

Consists of:

- Nodes – **nodeC**

diskC is a NSD defined over the internal disk of the node **nodeC** and is directly accessible only from site **C**

1. Create a dual-active ESS copy services domain assigning ESS A as **ServerA** and ESS B as **ServerB**. See the *IBM Enterprise Storage Server User's Guide*. In order to provide for the availability of at least one server after a disaster, the servers should reside at different sites.
2. Establish a PPRC logical path from LSS **A** to LSS **B**. See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
3. In order to protect the order of dependent writes that span multiple disk volumes, multiple LSS devices, or both, a consistency group should be defined over all logical subsystems at the primary site. See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*. In this case that would only be LSS A. See *Data integrity and the use of PPRC consistency groups*.
4. Establish a synchronous PPRC volume pair between the source and target using the **copy entire volume** option and leave the **permit read from secondary** option disabled. In this case it would be diskA-diskB. See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
5. Create a GPFS cluster defining the primary cluster configuration server as nodes **nodeA001** at site **A**, the secondary cluster configuration server as **nodeB001** at site **B**, an equal number of quorum nodes at each site, including the tiebreaker node at site **C**, **nodeC**. To prevent the tiebreaker node from assuming the role of file system manager, define it as **client**. Define all other quorum nodes as **manager**. List the nodes in the cluster in the file **NodeDescFile**. The **NodeDescFile** file contains the node descriptors:

   ```
   nodeA001:quourum-manager
   nodeA002:quorum-manager
   nodeA003:quorum-manager
   nodeA004:client
   nodeB001:quorum-manager
   nodeB002:quorum-manager
   nodeB003:quorum-manager
   nodeB004:client
   nodeC:quorum-client
   ```

   Issue this command:

   ```
   mmcrcluster –N NodeDescFile –p nodeA001 –s nodeB001
   ```
6. Enable the **unmountOnDiskFail** option on the tiebreaker node preventing false disk errors in the SAN configuration from being reported to the file system manager by issuing the **mmchconfig** command:

   ```
   mmchconfig unmountOnDiskFail=yes -N nodeC
   ```
7. Create an NSD over **diskA**. The disk descriptor contained in the file **DiskDescFile** is:

   ```
   /dev/diskA:nodeA001:nodeA002:dataAndMetadata:1
   ```

   Issue this command:

   ```
   mmcrnsd –F DiskDescFileP
   ```
8. Start the GPFS daemon on all nodes:

   ```
   mmstartup -a
   ```
9. Create a GPFS file system and mount it on all nodes at sites **A** and **B**.

   ```
   mmcrfs /gpfs/fs0 fs0 -F DiskDescFile
   ```

Failover to the recovery site and subsequent fail-back for an active/active configuration

For an active/active PPRC-based cluster, follow these steps to restore access to the file system through site **B** after site **A** has experienced a disastrous failure:

1. Stop the GPFS daemon on the surviving nodes as site **B** where the file **gpfs.siteB** lists all of the nodes at site **B**:

   ```
   mmshutdown -N gpfs.siteB
   ```

2. Perform PPRC failover, establishing a synchronous PPRC pair **diskB**-**diskA** with the PPRC failover option. See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*. The PPRC state of **diskB** is changed from *duplex target* to *suspended source* and is available for regular I/O. Refer to the *IBM Enterprise Storage Server* for a detailed explanation of PPRC failover.
3. If you needed to relax node quorum or make configuration changes, migrate the primary cluster configuration server to site **B**, issue this command:

   ```
   mmchcluster -p nodeB001
   ```
4. If site **C**, the tiebreaker, failed along with site **A**, existing node quorum designations must be relaxed in order to allow the surviving site to fulfill quorum requirements. To relax node quorum, temporarily change the designation of each of the failed quorum nodes to non-quorum nodes:

   ```
   mmchnode --nonquorum -N nodeA001,nodeA002,nodeA003,nodeC
   ```
5. Ensure the source volumes are *not* accessible to the recovery site:
   - Disconnect the cable
   - Define the **nsddevices** user exit file to exclude the source volumes
6. Restart the GPFS daemon on all surviving nodes:

   ```
   mmstartup -N gpfs.siteB
   ```

Once the operation of site **A** has been restored, the fail-back procedure is executed to restore the access to the file system from that location. The fail-back operation is a two-step process:

1. Resynchronize the paired volumes by establishing a temporary PPRC pair with **diskB** defined as the source and **diskA** as the target. The modification bitmap is traversed to find the mismatching disk sectors, whose content is then copied from **diskB** to **diskA**.
   a. Establish a PPRC path from LSS **B** to LSS **A** enabling the **consistency group** option. See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
   b. Establish a synchronous PPRC pair **diskB**-**diskA** with the PPRC **failback** option and **permit read from secondary** left disabled. See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
   c. Wait for the volume pair to reach the *duplex* (fully synchronized) state.
2. Shut GPFS down at site **B** and reverse the disk roles (the original primary disk becomes the primary again), bringing the PPRC pair to its initial state.
   a. Stop the GPFS daemon on all nodes.
   b. Establish a synchronous PPRC pair **diskA**-**diskB** with the PPRC failover option. See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
   c. Establish a synchronous PPRC pair **diskA**-**diskB** with the PPRC fail-back option and **permit read from secondary** left disabled. See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
   d. If during failover you migrated the primary cluster configuration server to a node in site **B**:
      1) Migrate the primary cluster configuration server back to site **A**:

         ```
         mmchcluster -p nodeA001
         ```
      2) Restore the initial quorum assignments:

         ```
         mmchnode --quorum -N nodeA001,nodeA002,nodeA003,nodeC
         ```
      3) Ensure that all nodes have the latest copy of the **mmfsdr** file:

         ```
         mmchcluster -p LATEST
         ```
   e. Ensure the source volumes *are* accessible to the recovery site:
      - Reconnect the cable
      - Edit the **nsddevices** user exit file to *include* the source volumes
   f. Start the GPFS daemon on all nodes:

      ```
      mmstartup -a
      ```

g. Mount the file system on all the nodes at sites **A** and **B**.

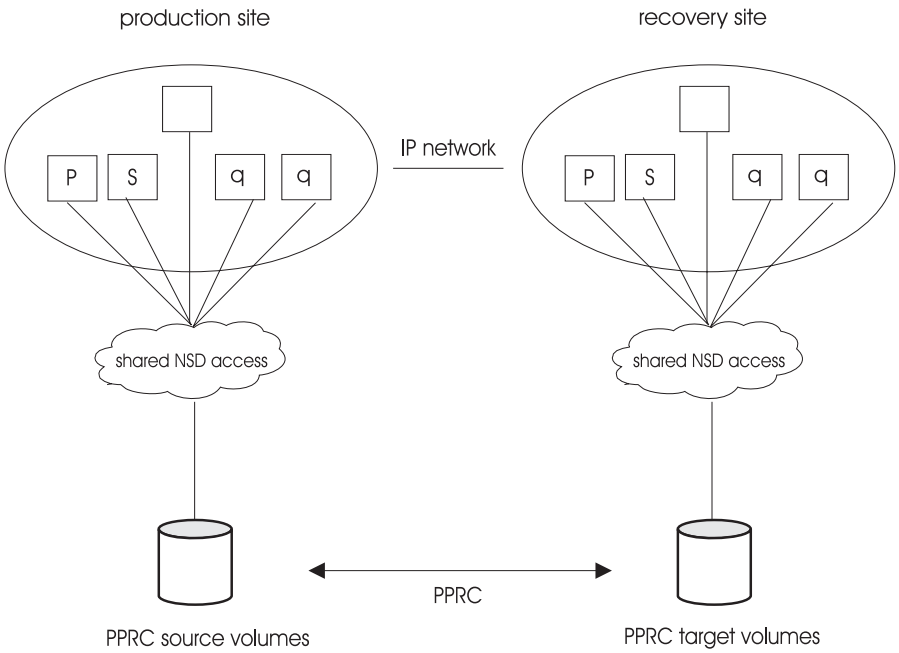
An active/passive GPFS cluster

In an active/passive environment, two GPFS clusters are set up in two geographically distinct locations (the production and the recovery sites). We refer to these as peer GPFS clusters. A GPFS file system is defined over a set of disk volumes located at the production site and these disks are mirrored using PPRC to a secondary set of volumes located at the recovery site. During normal operation, only the nodes in the production GPFS cluster mount and access the GPFS file system at any given time, which is the primary difference between a configuration of this type and the active/active model.

In the event of a catastrophe in the production cluster, the PPRC failover task is executed to enable access to the secondary replica of the file system located on the target PPRC disks. See IBM Redbooks® for *IBM TotalStorage Enterprise Storage Server Implementing ESS Copy Services in Open Environments* for a detailed explanation of PPRC failover and fail-back and the means of invoking these procedures in your environment.

The secondary replica is then mounted on nodes in the recovery cluster as a regular GPFS file system, thus allowing the processing of data to resume at the recovery site. At a latter point, after restoring the physical operation of the production site, we execute the fail-back procedure to resynchronize the content of the PPRC volume pairs between the two clusters and re-enable access to the file system in the production environment.

The high-level organization of synchronous active/passive PPRC-based GPFS cluster is shown in Figure 7.



P - primary cluster configuration server

S - secondary cluster configuration server

q - quorum node

Figure 7. A synchronous active/passive PPRC-based GPFS configuration without a tiebreaker site

Setting up an active/passive GPFS configuration

To establish an active/passive PPRC based GPFS cluster as shown in Figure 7 on page 60, consider the configuration:

Production site

Consists of:

- Nodes – **nodeP001**, **nodeP002**, **nodeP003**, **nodeP004**, **nodeP005**
- Storage subsystems – Enterprise Storage Server (ESS) **P**, logical subsystem (LSS) **P**
- LUN ids and disk volume names – **lunP1** (**hdisk11**), **lunP2** (**hdisk12**), **lunP3** (**hdisk13**), **lunP4** (**hdisk14**)

Recovery site

Consists of:

- Nodes – **nodeR001**, **nodeR002**, **nodeR003**, **nodeR004**, **nodeR005**
- Storage subsystems – ESS **R**, LSS **R**
- LUN ids and disk volume names – **lunR1** (**hdisk11**), **lunR2** (**hdisk12**), **lunR3** (**hdisk13**), **lunR4** (**hdisk14**)

All disks are SAN-attached and directly accessible from all local nodes.

1. Create a dual-active ESS copy services domain assigning ESS **P** as **ServerA** and ESS **R** as **ServerB**. See the *IBM Enterprise Storage Server User's Guide*.
2. Establish a PPRC logical path from LSS **P** to LSS **R** enabling the consistency group option. See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*, section on *Data integrity and the use of PPRC consistency groups*.
3. Establish synchronous PPRC volume pairs using the **copy entire volume** option and leave the **permit read from secondary** option disabled:

   ```
   lunP1-lunR1 (source-target)
   lunP2-lunR2 (source-target)
   lunP3-lunR3 (source-target)
   lunP4-lunR4 (source-target)
   ```

   See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
4. Create the recovery cluster selecting **nodeR001** as the primary cluster data server node, **nodeR002** as the secondary cluster data server nodes, and the nodes in the cluster contained in the file **NodeDescFileR**. The **NodeDescFileR** file contains the node descriptors:

   ```
   nodeR001:quourum-manager
   nodeR002:quourum-manager
   nodeR003:quourum-manager
   nodeR004:quourum-manager
   nodeR005
   ```

   Issue this command:

   ```
   mmcrcluster –N NodeDescFileR –p nodeR001 –s nodeR002
   ```
5. Create the GPFS production cluster selecting **nodeP001** as the primary cluster data server node, **nodeP002** as the secondary cluster data server node, and the nodes in the cluster contained in the file **NodeDescFileP**. The **NodeDescFileP** file contains the node descriptors:

   ```
   nodeP001:quourum-manager
   nodeP002:quourum-manager
   nodeP003:quourum-manager
   nodeP004:quourum-manager
   nodeP005
   ```

   Issue this command:

   ```
   mmcrcluster –N NodeDescFileP –p nodeP001 –s nodeP002
   ```

6. At all times the peer clusters must see a consistent image of the mirrored file system's configuration state contained in the **mmsdrfs** file. After the initial creation of the file system, all subsequent updates to the local configuration data must be propagated and imported into the peer cluster. Execute the **mmfsctl syncFSconfig** command to resynchronize the configuration state between the peer clusters after each of these actions in the primary GPFS cluster:
   - Addition of disks through the **mmadddisk** command
   - Removal of disks through the **mmdeldisk** command
   - Replacement of disks through the **mmrpldisk** command
   - Modifications to disk attributes through the **mmchdisk** command
   - Changes to the file system's mount point through the **mmchfs -T** command

   To automate the propagation of the configuration state to the recovery cluster, activate and use the **syncFSconfig** user exit. Follow the instructions in the prolog of **/usr/lpp/mmfs/samples/syncfsconfig.sample**.
7. From a node in the production cluster, start the GPFS daemon on all nodes:
   ```
   mmstartup -a
   ```
8. Create the NSDs at the production site. The disk descriptors contained in the file **DiskDescFileP** are:
   ```
   /dev/hdisk11:nodeP001:nodeP002:dataAndMetadata:-1
   /dev/hdisk12:nodeP001:nodeP002:dataAndMetadata:-1
   /dev/hdisk13:nodeP001:nodeP002:dataAndMetadata:-1
   /dev/hdisk14:nodeP001:nodeP002:dataAndMetadata:-1
   ```

   Issue this command:
   ```
   mmcrnsd –F DiskDescFileP
   ```
9. Create the GPFS file system and mount it on all nodes at the production site:
   ```
   mmcrfs /gpfs/fs0 fs0 -F DiskDescFileP
   ```

Failover to the recovery site and subsequent fail-back for an active/passive configuration

For an active/passive PPRC-based cluster, follow these steps to failover production to the recovery site:

1. If the GPFS daemon is not already stopped on all surviving nodes in the production cluster, from a node in the production cluster issue:
   ```
   mmshutdown –a
   ```
2. Perform PPRC failover. This step involves establishing synchronous PPRC pairs with the failover option:
   - **lunR1-lunP1**
   - **lunR2-lunP2**
   - **lunR3-lunP3**
   - **lunR4-lunP4**

   See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*. After the completion of this step, the volumes on ESS R enter the **suspended source** PPRC state.
3. From a node in the recovery cluster start GPFS:
   ```
   mmstartup –a
   ```
4. Mount the file system on all nodes in the recovery cluster.

Once the physical operation of the production site has been restored, execute the fail-back procedure to transfer the file system activity back to the production GPFS cluster. The fail-back operation is a two-step process:

1. For each of the paired volumes, resynchronize the pairs by establishing a temporary PPRC pair with the recovery LUN acting as the sources for the production LUN. The PPRC modification bitmap is traversed to find the mismatching disk tracks, whose content is then copied from the recovery LUN to the production LUN.
   a. Establish a PPRC path from LSS **R** to LSS **P** enabling the **consistency group** option. See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
   b. Establish synchronous PPRC volume pairs with the **failback** option, leaving the **permit read from secondary** option disabled.
      - **lunR1**-**lunP1**
      - **lunR2**-**lunP2**
      - **lunR3**-**lunP3**
      - **lunR4**-**lunP4**

      See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
   c. Wait for all PPRC pairs to reach the **duplex** (fully synchronized) state.
   d. If the state of the system configuration has changed, update the GPFS configuration data in the production cluster to propagate the changes made while in failover mode. From a node at the recovery site, issue:

      ```
      mmfsctl all syncFSconfig –n gpfs.sitePnodes
      ```
2. Stop GPFS on all nodes in the recovery cluster and reverse the disk roles so the original primary disks become the primaries again:
   a. From a node in the recovery cluster, stop the GPFS daemon on all nodes in the recovery cluster:

      ```
      mmshutdown –a
      ```
   b. Establish synchronous PPRC volume pairs with the PPRC failover option:
      - **lunP1**-**lunR1**
      - **lunP2**-**lunR2**
      - **lunP3**-**lunR3**
      - **lunP4**-**lunR4**

      See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
   c. Establish synchronous PPRC volume pairs with the PPRC failback option, leaving the permit **read from secondary** option disabled:
      - **lunP1**-**lunR1**
      - **lunP2**-**lunR2**
      - **lunP3**-**lunR3**
      - **lunP4**-**lunR4**

      See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
   d. From a node in the production cluster, start GPFS:

      ```
      mmstartup –a
      ```
   e. From a node in the production cluster, mount the file system on all nodes in the production cluster.

Data integrity and the use of PPRC consistency groups

The integrity of the post-disaster replica of the file system contained on the secondary PPRC disk volumes depends on the assumption that the order of dependent write operations is preserved by the PPRC mirroring mechanism. While the synchronous nature of PPRC guarantees such ordering during periods of stability, certain types of rolling failure scenarios require additional consideration. By default, without the PPRC **consistency group** option enabled, if the primary ESS detects the loss of the physical PPRC connectivity or the failure of one of the secondary disks, it responds by moving the corresponding primary volumes to the suspended state but continues to process the subsequent I/O requests as normal. The subsystem does not report this condition to the driver and, as a result, the application continues normal

processing without any knowledge of the lost updates. GPFS relies on log recovery techniques to provide for the atomicity of updates to the file system's metadata, which is why such behavior would expose GPFS to a serious data integrity risk. Therefore to provide for the proper ordering of updates to the recovery copy of the file system, it is suggested you always enable the **consistency group** option when configuring the PPRC paths between the peer logical subsystems.

Figure 8 illustrates this problem. Consider a setup with two primary and two secondary subsystems, each providing access to two LUNs. The four primary LUNs make up the primary replica of the file system. At some point, GPFS attempts to issue a sequence of four write requests, one to each primary disk, with the expectation that the updates appear in the exact order they were issued. If PPRC path 1 breaks before the start of the first write, the recovery site receives updates 3 and 4, but not necessarily 1 and 2 – a result that violates the write dependency rule and renders the target replica of the file system unusable.

The PPRC **consistency group** option determines the behavior of the primary subsystem in situations where a sudden failure of the secondary volume, or a failure of the inter-site interconnect, makes it impossible to sustain the normal synchronous mirroring process. If the PPRC path between the pair has been defined with the **consistency group** option, the primary volume enters a long busy state, and the subsystem reports this condition back to the host system with the QUEUE FULL (QF) SCSI status byte code. Typically, the driver makes several attempts to re-queue the request and ultimately reports the failure to the requestor. This allows GPFS to execute the appropriate action in response to the failure by either marking the disk down or panicking the file system. See the *General Parallel File System: Problem Determination Guide*.

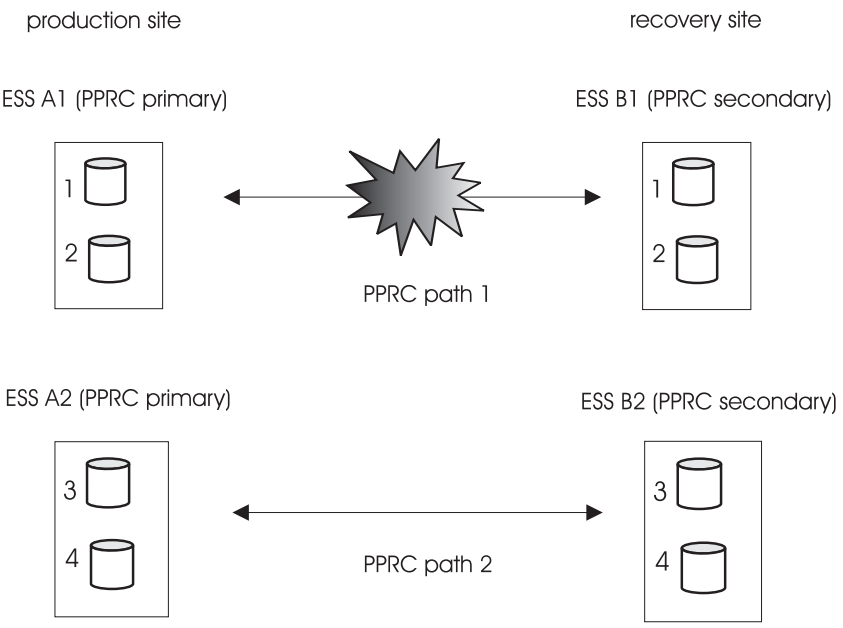


Figure 8. Violation of write ordering without the use of a PPRC consistency group

Asynchronous mirroring utilizing ESS FlashCopy

The FlashCopy feature of ESS provides an easy implementation to make a point-in-time copy of a GPFS file system as an online backup mechanism. This function provides an instantaneous copy of the original data on the target disk, while the actual transfer of data takes place asynchronously and is fully transparent to the user.

When a FlashCopy disk is first created, the subsystem establishes a control bitmap that is subsequently used to track the changes between the source and the target disks. When processing read I/O requests sent to the target disk, this bitmap is consulted to determine whether the request can be satisfied using the

target's copy of the requested block. If the track containing the requested data has not yet been copied, the source disk is instead accessed and its copy of the data is used to satisfy the request. The FlashCopy feature is further described in the *IBM Enterprise Storage Server*. Similar to PPRC, FlashCopy operations can be invoked using the ESS Copy Services web interface or the command-line scripts.

To prevent the appearance of out-of-order updates, it is important to consider data consistency when using FlashCopy. Prior to taking the FlashCopy image all disk volumes that make up the file system must be brought to same logical point in time. Two methods may be used to provide for data consistency in the FlashCopy image of your GPFS file system. Both techniques guarantee the consistency of the FlashCopy image by the means of temporary suspension of I/O, but either can be seen as the preferred method depending on your specific requirements and the nature of your GPFS client application:

- The use of FlashCopy consistency groups provides for the proper ordering of updates, but this method does not by itself suffice to guarantee the atomicity of updates as seen from the point of view of the user application. If the application process is actively writing data to GPFS, the on-disk content of the file system may, at any point in time, contain some number of incomplete data record updates and possibly some number of in-progress updates to the GPFS metadata. These appear as partial updates in the FlashCopy image of the file system, which must be dealt before enabling the image for normal file system use. The use of metadata logging techniques enables GPFS to detect and recover from these partial updates to the file system's metadata. However, ensuring the atomicity of updates to the actual data remains the responsibility of the user application. Consequently, the use of FlashCopy consistency groups is suitable only for applications that implement proper mechanisms for the recovery from incomplete updates to their data.

  The FlashCopy consistency group mechanism is used to *freeze* the source disk volume at the logical instant at which its image appears on the target disk:

  1. Issue the FlashCopy creation command, enabling the **freeze FlashCopy consistency group** option to suspend all I/O activity against each source volume in the file system. All I/O requests directed to the volumes now receive the SCSI status code **QUEUE FULL** (QF).
  2. Issue the **FlashCopy consistency created** task to release the consistency group and resume the normal processing of I/O. For details on the use of FlashCopy consistency groups, see the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.

  Assuming a configuration with:

  – Storage subsystems – ESS 1; logical subsystem LSS 1

  – LUN ids and disk volume names – **lunS1** (**hdisk11**), **lunS2** (**hdisk12**), **lunT1**, **lunT2**

    lunS1 and **lunS2** are the FlashCopy source volumes. These disks are SAN-connected and appear on the GPFS nodes as **hdisk11** and **hdisk12**, respectively. A single GPFS file system **fs0** has been defined over these two disks.

    lunT1 and **lunT2** are the FlashCopy target volumes. None of the GPFS nodes have direct connectivity to these disks.

  To generate a FlashCopy image using a consistency group:

  1. Run the **establish FlashCopy pair** task with the **freeze FlashCopy consistency group** option. Create the volume pairs:

     ```
     lunS1 – lunT1  (source-target)
     lunS2 – lunT2  (source-target)
     ```

     See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
  2. Run the **consistency created** task on all logical subsystems that hold any of the FlashCopy disks. See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
- The use of file-system-level suspension through the **mmfsctl** command prevents incomplete updates in the FlashCopy image and is the suggested method for protecting the integrity of your FlashCopy images. Issuing the **mmfsctl** command leaves the on-disk copy of the file system in a fully consistent state, ready to be flashed and copied onto a set of backup disks. The command instructs GPFS to flush the data buffers on all nodes, write the cached metadata structures to disk, and suspend the execution of all subsequent I/O requests.

1. To initiate file-system-level suspension, issue the **mmfsctl suspend** command.
2. To resume normal file system I/O, issue the **mmfsctl resume** command.

Assuming a configuration with:

– Storage subsystems – ESS 1; logical subsystem LSS 1

– LUN ids and disk volume names – **lunS1** (**hdisk11**), **lunS2** (**hdisk12**), **lunT1**, **lunT2**

  lunS1 and **lunS2** are the FlashCopy source volumes. These disks are SAN-connected and appear on the GPFS nodes as **hdisk11** and **hdisk12**, respectively. A single GPFS file system **fs0** has been defined over these two disks.

  lunT1 and **lunT2** are the FlashCopy target volumes. None of the GPFS nodes have direct connectivity to these disks.

To generate a FlashCopy image using file-system-level suspension:

1. From any node in the GPFS cluster, suspend all file system activity and flush the GPFS buffers on all nodes:

   ```
   mmfsctl fs0 suspend
   ```

2. Run the **establish FlashCopy pair** task to create the following volume pairs:

   ```
   lunS1 – lunT1  (source-target)
   lunS2 – lunT2  (source-target)
   ```

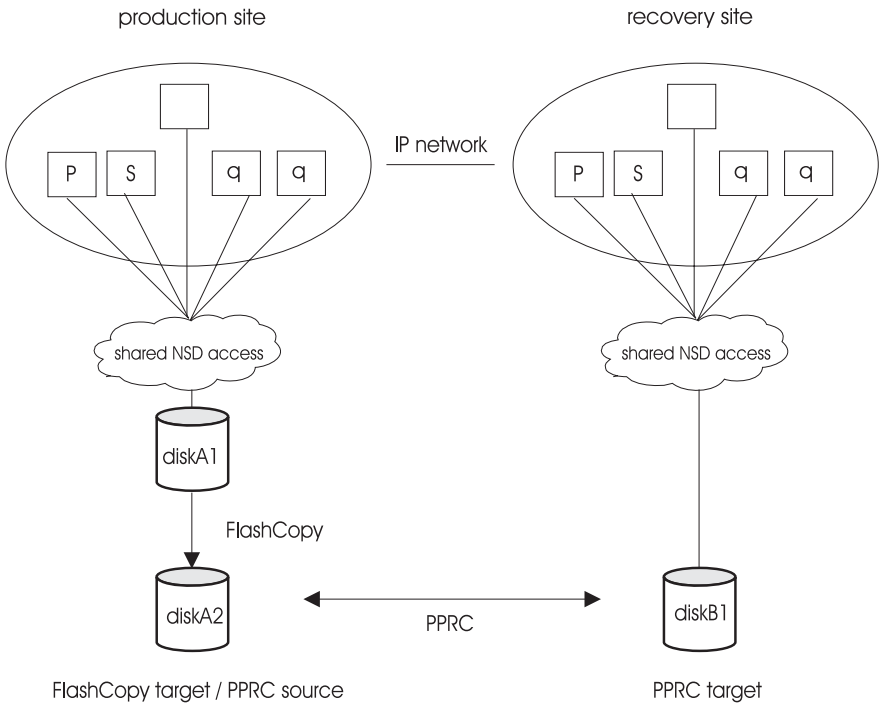
   See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.

3. From any node in the GPFS cluster, resume the file system activity:

   ```
   mmfsctl fs0 resume
   ```

Several uses of the FlashCopy replica after its initial creation can be considered. For example, if your primary operating environment suffers a permanent loss or a corruption of data, you may choose to flash the target disks back onto the originals to quickly restore access to a copy of the file system as seen at the time of the previous snapshot. Before restoring the file system from a FlashCopy, please make sure to suspend the activity of the GPFS client processes and unmount the file system on all GPFS nodes.

To protect your data against site-wide disasters, you may chose to instead transfer the replica offsite to a remote location using PPRC or any other type of bulk data transfer technology. Alternatively, you may choose to mount the FlashCopy image as a separate file system on your backup processing server and transfer the data to tape storage. To enable regular file system access to your FlashCopy replica, create a single-node GPFS cluster on your backup server node and execute the **mmfsctl syncFSconfig** command to import the definition of the file system from your production GPFS cluster. Figure 9 on page 67 provides a schematic view of an asynchronous recovery GPFS environment using a combination of PPRC and FlashCopy. In such environments, two independent GPFS clusters are set up in distinct geographic locations (production and recovery sites). We refer to such clusters as *peer* GPFS clusters. Peer clusters must share the same UID/GID space, but otherwise need not belong to the same administrative domain. In particular, the administrator is free to identify nodes in both clusters with the same set of short hostnames if such a configuration is indeed desired.



P - primary cluster configuration server

S - secondary cluster configuration server

q - quorum node

Figure 9. High-level organization of a FlashCopy/PPRC recovery environment

FlashCopy/PPRC provides for the availability of the file system's on-disk content in the recovery cluster. But in order to make the file system known and accessible, you must issue the **mmfsctl syncFSConfig** command to:

- Import the state of the file system's configuration from the primary location.
- Propagate all relevant changes to the configuration in the primary cluster to its peer to prevent the risks of discrepancy between the peer's **mmsdrfs** file and the content of the file system descriptor found in the snapshot.

  It is suggested you generate a new FlashCopy replica immediately after every administrative change to the state of the file system. This eliminates the risk of a discrepancy between the GPFS configuration data contained in the **mmsdrfs** file and the on-disk content of the replica.

Restriction: The primary copy of a GPFS file system and its FlashCopy image cannot coexist in the same GPFS cluster. A node can mount either the original copy of the file system or one of its FlashCopy replicas, but not both. This restriction has to do with the current implementation of the NSD-to-LUN mapping mechanism, which scans all locally-attached disks, searching for a specific value (the NSD id) at a particular location on disk. If both the original volume and its FlashCopy image are visible to a particular node, these disks would appear to GPFS as distinct devices with identical NSD ids

For this reason, we ask users to zone their SAN configurations such that at most one replica of any given GPFS disk is visible from any node. That is, the nodes in your production cluster should have access to the disks that make up the actual file system but should not see the disks holding the FlashCopy images, whereas the backup server should see the FlashCopy targets but not the originals.

Alternatively, you can use the **nsddevices** user exit located in **/var/mmfs/etc/** to explicitly define the subset of the locally visible disks to be accessed during the NSD device scan on the local node. See "Setting up FlashCopy using file-system-level suspension."

In the production GPFS cluster, FlashCopy is used to take periodic volume-level snapshots of the GPFS file system onto a set of idle local disks and the snapshots are then propagated to the peer recovery cluster using PPRC. The goal of this technique is to provide a consistent (but not necessarily up-to-date) image of the file system at the recovery site that can be used to restore operation in the event of a disaster in the primary cluster. Note that since from the GPFS perspective, the replicas are two entirely distinct file systems, nothing prevents the administrator from mounting and operating on both replicas concurrently if deemed necessary.

Setting up FlashCopy using file-system-level suspension

To prepare a file system as depicted in Figure 9 on page 67, using file-system-level suspension to provide for data consistency, consider the configuration:

Site A - primary cluster
Consisting of:

- Nodes – **nodeA001**, **nodeA002**, **nodeA003**, **nodeA004**, **nodeA005**
- Disk device names – **diskA1**, **diskA2**

Site B - recovery site
Consisting of:

- Nodes – **nodeB001**, **nodeB002**, **nodeB003**, **nodeB004**, **nodeB005**
- Disks – **diskB1**

There is a single file system, **fs0**, defined on **diskA1**. To create a volume-level snapshot and propagate the snapshot to the recovery cluster:

1. Define an **nsddevices** user exit file to prevent the production site from using the FlashCopy target disk **diskA2**:

   ```
   echo "echo diskA1 hdisk" > /var/mmfs/etc/nsddevices
   chmod 744 /var/mmfs/etc/nsddevices
   ```

   Refer to the prolog of **/usr/lpp/mmfs/samples/nsddevices.samples** for detailed instructions on the usage of **nsddevices**.
2. In the primary cluster, suspend all file system I/O activity and flush the GPFS buffers:

   ```
   mmfsctl fs0 suspend
   ```
3. Establish a FlashCopy pair using **diskA1** as the source and **diskA2** as the target. See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
4. Resume the file system I/O activity:

   ```
   mmfsctl fs0 resume
   ```
5. Establish a PPRC path and a synchronous PPRC volume pair **diskA2**-**diskB** (primary-secondary). Use the **copy entire volume** option and leave the **permit read from secondary** option disabled. See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
6. Wait for the completion of the FlashCopy background task. Wait for the PPRC pair to reach the *duplex* (fully synchronized) state.
7. Terminate the PPRC volume pair **diskA2**-**diskB**. See the *IBM Enterprise Storage Server Implementing ESS Copy Services in Open Environments*.
8. If this is the first time the snapshot is taken, or if the configuration state of **fs0** changed since the previous snapshot, propagate the most recent configuration to site **B**:

   ```
   mmfsctl all syncFSconfig -n nodes.siteB
   ```

The file **nodes.siteB** lists all of the nodes, one per line, at the recovery site.

Chapter 5. Implementing a clustered NFS using GPFS on Linux

In addition to the traditional exporting of GPFS file systems using the Network File System (NFS) protocol, GPFS allows you to configure a subset of the nodes in the cluster to provide a highly-available solution for exporting GPFS file systems using NFS.

The participating nodes are designated as Cluster NFS (CNFS) member nodes and the entire setup is frequently referred to as CNFS or a CNFS cluster.

In this solution, all CNFS nodes export the same file systems to the NFS clients. When one of the CNFS nodes fails, the NFS serving load moves from the failing node to another node in the CNFS cluster. Failover is done using recovery groups to help choose the preferred node for takeover.

Currently, CNFS is supported only in the Linux environment. For an up-to-date list of supported operating systems, specific distributions, and other dependencies, refer to the GPFS FAQ at: publib.boulder.ibm.com/infocenter/clresctr/topic/com.ibm.cluster.gpfs.doc/gpfs_faqs/gpfsclustersfaq.html.

NFS monitoring

Every node in the CNFS cluster runs an NFS utility that monitors GPFS, NFS, and networking components on the node. Upon failure detection and based on your configuration, the monitoring utility might invoke a failover.

NFS failover

As part of GPFS recovery, the CNFS cluster failover mechanism is invoked. It transfers the NFS serving load that was served by the failing node to another node in the CNFS cluster. Failover is done using recovery groups to help choose the preferred node for takeover.

The failover mechanism is based on IP address failover. In addition, it guarantees NFS lock (NLM) recovery.

NFS locking and load balancing

CNFS supports a failover of all of the node's load together (all of its NFS IP addresses) as one unit to another node. However, if no locks are outstanding, individual IP addresses can be moved to other nodes for load balancing purposes.

CNFS is based on round robin DNS for load balancing of NFS clients among the NFS cluster nodes.

CNFS network setup

In addition to one set of IP addresses for the GPFS cluster, a separate set of one or more IP addresses is required for NFS serving.

The NFS IP addresses can be real or virtual (aliased). These addresses *must* be configured to be static (not DHCP) and to not start at boot time. Load balancing is achieved using a round robin DNS. NFS clients are distributed among the NFS cluster nodes at mount time.

CNFS setup

You can set up a clustered NFS environment within a GPFS cluster.

To do this, follow these steps:

1. Designate a separate directory for the CNFS shared files:

   mmchconfig cnfsSharedRoot=*directory*

   where:

   cnfsSharedRoot=*directory*
   Is the path name to a GPFS directory, preferably on a small separate file system that is not exported by NFS. The GPFS file system that contains the directory must be configured to be mounted automatically upon GPFS start on all of the CNFS nodes (**-A yes** option on the **mmchfs** command). **cnfsSharedRoot** is a mandatory parameter and must be defined first.
2. Add all GPFS file systems that need to be exported to **/etc/exports**. See ″Exporting a GPFS file system using NFS″ in the *GPFS: Administration and Programming Reference* for NFS export considerations. If the shared directory from step 1 is in an exported file system, restrict access to that directory.
3. Use the **mmchnode** command to add nodes to the CNFS cluster:

   mmchnode --cnfs-interface=*nfs_ip* **-N** *node*

   where:

   nfs_ip Is a comma-separated list of virtual IP addresses (VIP). There is typically one VIP address.

   node Identifies a GPFS node to be added to the CNFS cluster.

   See the description of the **mmchnode** command in the *GPFS: Administration and Programming Reference* for information about how to use the command with a list of nodes.
4. Use the **mmchconfig** command to configure the optional CNFS parameters. You can specify one or more of the following:

   cnfsMountdPort=*mountd_port*
   Specifies the port number to be used for the **rpc.mountd** daemon.

   For CNFS to work correctly with the automounter (AMD), the **rpc.mountd** daemon on the different nodes must be bound to the same port.

   cnfsNFSDprocs=*nfsd_procs*
   Specifies the number of **nfsd** kernel threads. The default is 32.

   cnfsVIP=*dns_name*
   Specifies a virtual DNS name for the list of CNFS IP addresses that are assigned to the nodes with the **mmchnode** command (step 3). This allows NFS clients to be distributed among the CNFS nodes using round robin DNS.
5. If multiple failover groups are desired, assign a group ID to each NFS node:

   mmchnode --cnfs-groupid=*nn* **-N** *node*

   To assign NFS nodes to different groups, use a group ID that is in a different range of ten. For example, a node with group ID 2*n* will fail over only to nodes in the same group (which means any node with group ID 20 to 29). Failover in the same group will first look for one of the nodes with the same group ID. If none are found, any node in the group range starting at *n*0 to *n*9 is selected.

CNFS administration

There are some common CNFS administration tasks in this topic along with a sample configuration.

To query the current CNFS configuration, enter:

mmlscluster --cnfs

To temporarily disable CNFS on one or more nodes, enter:

mmchnode --cnfs-disable -N *NodeList*

Note: This operation affects only the high-availability aspects of the CNFS functionality. Normal NFS exporting of the data from the node is not affected. There will be no automatic failover from or to this node in case of a failure.

To re-enable previously-disabled CNFS member nodes, enter:

mmchnode --cnfs-enable -N *NodeList*

To remove nodes from the CNFS cluster, enter:

mmchnode --cnfs-interface=DELETE -N *NodeList*

Note: This operation affects only the high-availability aspects of the CNFS functionality. Normal NFS exporting of the data from the node is not affected. There will be no automatic failover from or to this node in case of a failure.

A sample CNFS configuration

Here is a CNFS configuration example, which assumes the following:

- Your GPFS cluster contains 3 nodes: fin18, fin19, and fin20
- The virtual IP addressess for NFS serving are: fin18nfs, fin19nfs, and fin20nfs
- The DNS name used for load balancing NFS is: finnfs

To define a CNFS cluster made up of these nodes, follow these steps:

1. Add the desired GPFS file systems to **/etc/exports** on each of the nodes.
2. Create a directory called **ha** in one of the GPFS file systems by entering:

   ```
   mkdir /gpfs/fs1/ha
   ```

3. Create a temporary file called **/tmp/hanfs-list**, which contains the following lines:

   ```
   fin18 --cnfs-interface=fin18nfs
   fin19 --cnfs-interface=fin19nfs
   fin20 --cnfs-interface=fin20nfs
   ```

4. Set the CNFS shared directory and cluster virtual IP address, by entering:

   ```
   mmchconfig cnfsSharedRoot=/gpfs/fs1/ha,cnfsVIP=finnfs
   ```

5. Create the CNFS cluster with the **mmchnode** command, by entering:

   ```
   mmchnode -S /tmp/hanfs-list
   ```

6. Access the exported GPFS file systems over NFS. If one or more GPFS nodes fail, the NFS clients should continue uninterrupted.

Chapter 6. Monitoring GPFS I/O performance with the mmpmon command

Use the **mmpmon** command to monitor GPFS performance on the node in which it is run, and other specified nodes.

Before attempting to use the **mmpmon** command, review the command documentation in the *General Parallel File System: Administration and Programming Reference*, and then read this entire chapter.

These are the topics related to **mmpmon**:

Overview of mmpmon

The **mmpmon** facility allows the system administrator to collect I/O statistics from the point of view of GPFS servicing application I/O requests.

The collected data can be used for many purposes, including:

- Tracking IO demand over longer periods of time - weeks or months.
- Recording I/O patterns over time (when peak usage occurs, and so forth).
- Determining if some nodes service more application demand than others.
- Monitoring the I/O patterns of a single application which is spread across multiple nodes.
- Recording application I/O request service times.

Figure 10 shows the software layers in a typical system with GPFS. **mmpmon** is built into GPFS.

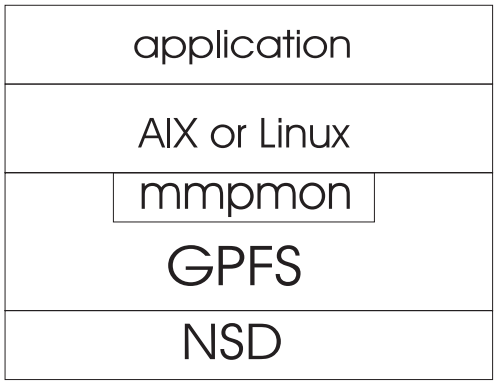


Figure 10. Node running mmpmon

Specifying input to the mmpmon command

The **mmpmon** command must be run using root authority. For command syntax, see **mmpmon** in the *General Parallel File System: Administration and Programming Reference*.

The **mmpmon** command is controlled by an input file that contains a series of requests, one per line. This input can be specified with the **-i** flag, or read from standard input (stdin). Providing input using stdin allows **mmpmon** to take keyboard input or output piped from a user script or application.

Leading blanks in the input file are ignored. A line beginning with a pound sign (#) is treated as a comment. Leading blanks in a line whose first non-blank character is a pound sign (#) are ignored.

Table 3 describes the **mmpmon** input requests.

Table 3. Input requests to the **mmpmon** *command*

Request	Description
fs_io_s	"Display I/O statistics per mounted file system"
io_s	"Display I/O statistics for the entire node" on page 78
nlist add *name*[*name*...]	"Add node names to a list of nodes for mmpmon processing" on page 80
nlist del	"Delete a node list" on page 82
nlist new *name*[*name*...]	"Create a new node list" on page 82
nlist s	"Show the contents of the current node list" on page 83
nlist sub *name*[*name*...]	"Delete node names from a list of nodes for mmpmon processing" on page 84
once *request*	Indicates that the request is to be performed only once.
reset	"Reset statistics to zero" on page 79
rhist nr	"Changing the request histogram facility request size and latency ranges" on page 89
rhist off	"Disabling the request histogram facility" on page 91. This is the default.
rhist on	"Enabling the request histogram facility" on page 92
rhist p	"Displaying the request histogram facility pattern" on page 92
rhist reset	"Resetting the request histogram facility data to zero" on page 95
rhist s	"Displaying the request histogram facility statistics values" on page 96
source *filename*	"Using request *source* and prefix directive *once*" on page 101
ver	"Displaying mmpmon version" on page 98

Running mmpmon on multiple nodes

The **mmpmon** command may be invoked on one node to submit requests to multiple nodes in a local GPFS cluster by using the **nlist** requests. See "Understanding the node list facility" on page 80.

Running mmpmon concurrently from multiple users on the same node

Five instances of **mmpmon** may be run on a given node concurrently. This is intended primarily to allow different user-written performance analysis applications or scripts to work with the performance data. For example, one analysis application might deal with **fs_io_s** and **io_s** data, while another one deals with **rhist** data, and another gathers data from other nodes in the cluster. The applications might be separately written or separately maintained, or have different sleep and wake-up schedules.

Be aware that there is only one set of counters for **fs_io_s** and **io_s** data, and another, separate set for **rhist** data. Multiple analysis applications dealing with the same set of data must coordinate any activities that could reset the counters, or in the case of **rhist** requests, disable the feature or modify the ranges.

Display I/O statistics per mounted file system

The **fs_io_s** (file system I/O statistics) request returns strings containing I/O statistics taken over all mounted file systems as seen by that node, and are presented as total values for each file system. The values are cumulative since the file systems were mounted or since the last **reset** request, whichever is most recent. When a file system is unmounted, its statistics are lost.

Read and write statistics are recorded separately. The statistics for a given file system are for the file system activity on the node running **mmpmon**, not the file system in total (across the cluster).

Table 4 describes the keywords for the **fs_io_s** response, in the order that they appear in the output. These keywords are used only when **mmpmon** is invoked with the **-p** flag.

Table 4. Keywords and values for the **mmpmon fs_io_s** *response*

Keyword	Description
_n_	IP address of the node responding. This is the address by which GPFS knows the node.
_nn_	hostname that corresponds to the above IP address.
_rc_	Indicates the status of the operation.
_t_	Current time of day in seconds (absolute seconds since Epoch (1970)).
_tu_	Microseconds part of the current time of day.
_cl_	Name of the cluster that owns the file system.
_fs_	The name of the file system for which data are being presented.
_d_	The number of disks in the file system.
_br_	Total number of bytes read, from both disk and cache.
_bw_	Total number of bytes written, to both disk and cache.
_oc_	Count of **open()** call requests serviced by GPFS. This also includes **creat()** call counts.
_cc_	Number of **close()** call requests serviced by GPFS.
rdc	Number of application read requests serviced by GPFS.
_wc_	Number of application write requests serviced by GPFS.
dir	Number of **readdir()** call requests serviced by GPFS.
_iu_	Number of inode updates to disk.

Example of mmpmon fs_io_s request

Assume that **commandFile** contains this line:

```
fs_io_s
```

and this command is issued:

```
mmpmon -p  -i commandFile
```

The output is two lines in total, and similar to this:

```
_fs_io_s_ _n_ 1992.18.1.8 _nn_ node1 _rc_ 0 _t_ 1066660148 _tu_ 407431 _cl_ myCluster.xxx.com
_fs_ gpfs2 _d_ 2 _br_ 6291456 _bw_ 314572800 _oc_ 10 _cc_ 16 _rdc_ 101 _wc_ 300 _dir_ 7 _iu_ 2
_fs_io_s_ _n_ 199.18.1.8 _nn_ node1 _rc_ 0 _t_ 1066660148 _tu_ 407455 _cl_ myCluster.xxx.com
_fs_ gpfs1 _d_ 3 _br_ 5431636 _bw_ 173342800 _oc_ 6 _cc_ 8 _rdc_ 54 _wc_ 156 _dir_ 3 _iu_ 6
```

The output consists of one string per mounted file system. In this example, there are two mounted file systems, **gpfs1** and **gpfs2**.

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.8 name node1 fs_io_s OK
cluster:    myCluster.xxx.com
filesystem: gpfs2
disks: 2
timestamp: 1066660148/407431
bytes read: 6291456
bytes written: 314572800
opens: 10
```

```
closes: 16
reads: 101
writes: 300
readdir: 7
inode updates: 2

mmpmon node 199.18.1.8 name node1 fs_io_s OK
cluster:    myCluster.xxx.com
filesystem: gpfs1
disks: 3
timestamp: 1066660148/407455
bytes read: 5431636
bytes written: 173342800
opens: 6
closes: 8
reads: 54
writes: 156
readdir: 3
inode updates: 6
```

When no file systems are mounted, the responses are similar to:

```
_fs_io_s_ _n_ 199.18.1.8 _nn_ node1 _rc_ 1 _t_ 1066660148 _tu_ 407431 _cl_ - _fs_ -
```

The **_rc_** field is nonzero and the both the **_fs_** and **_cl_** fields contains a minus sign. If the **-p** flag is not specified, the results are similar to:

```
mmpmon node 199.18.1.8 name node1 fs_io_s status 1
no file systems mounted
```

Display I/O statistics for the entire node

The **io_s** (I/O statistics) request returns strings containing I/O statistics taken over all mounted file systems as seen by that node, and are presented as total values for the entire node. The values are cumulative since the file systems were mounted or since the last **reset**, whichever is most recent. When a file system is unmounted, its statistics are lost and its contribution to the total node statistics vanishes. Read and write statistics are recorded separately.

Table 5 describes the keywords for the **io_s** response, in the order that they appear in the output. These keywords are used only when **mmpmon** is invoked with the **-p** flag.

Table 5. Keywords and values for the **mmpmon io_s** *response*

Keyword	Description
_n_	IP address of the node responding. This is the address by which GPFS knows the node.
_nn_	hostname that corresponds to the above IP address.
_rc_	Indicates the status of the operation.
_t_	Current time of day in seconds (absolute seconds since Epoch (1970)).
_tu_	Microseconds part of the current time of day.
_br_	Total number of bytes read, from both disk and cache.
_bw_	Total number of bytes written, to both disk and cache.
_oc_	Count of **open()** call requests serviced by GPFS. The open count also includes **creat()** call counts.
_cc_	Number of **close()** call requests serviced by GPFS.
rdc	Number of application read requests serviced by GPFS.
_wc_	Number of application write requests serviced by GPFS.
dir	Number of **readdir()** call requests serviced by GPFS.

Table 5. Keywords and values for the **mmpmon io_s** *response (continued)*

Keyword	Description
_iu_	Number of inode updates to disk. This includes inodes flushed to disk because of access time updates.

Example of mmpmon io_s request

Assume that **commandFile** contains this line:

```
io_s
```

and this command is issued:

```
mmpmon -p -i commandFile
```

The output is one line in total, and similar to this:

```
_io_s_ _n_ 199.18.1.8 _nn_ node1 _rc_ 0 _t_ 1066660148 _tu_ 407431 _br_ 6291456
_bw_ 314572800 _oc_ 10 _cc_ 16 _rdc_ 101 _wc_ 300 _dir_ 7 _iu_ 2
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.8 name node1 io_s OK
timestamp: 1066660148/407431
bytes read: 6291456
bytes written: 314572800
opens: 10
closes: 16
reads: 101
writes: 300
readdir: 7
inode updates: 2
```

Reset statistics to zero

The **reset** request resets the statistics that are displayed with **fs_io_s** and **io_s** requests. The **reset** request *does not* reset the histogram data, which is controlled and displayed with **rhist** requests. Table 6 describes the keywords for the **reset** response, in the order that they appear in the output. These keywords are used only when **mmpmon** is invoked with the **-p** flag. The response is a single string.

Table 6. Keywords and values for the **mmpmon reset** *response*

Keyword	Description
_n_	IP address of the node responding. This is the address by which GPFS knows the node.
_nn_	hostname that corresponds to the above IP address.
_rc_	Indicates the status of the operation.
_t_	Current time of day in seconds (absolute seconds since Epoch (1970)).
_tu_	Microseconds part of the current time of day.

Example of mmpmon reset request

Assume that **commandFile** contains this line:

```
reset
```

and this command is issued:

```
mmpmon -p -i commandFile
```

The output is similar to this:

```
_reset_ _n_ 199.18.1.8 _nn_ node1 _rc_ 0 _t_ 1066660148 _tu_ 407431
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.8 name node1 reset OK
```

Understanding the node list facility

The node list facility can be used to invoke **mmpmon** on multiple nodes and gather data from other nodes in the cluster. Table 7 describes the **nlist** requests:

Table 7. nlist requests for the **mmpmon** *command*

Request	Description
nlist add *name*[*name*...]	"Add node names to a list of nodes for mmpmon processing"
nlist del	"Delete a node list" on page 82
nlist new *name*[*name*...]	"Create a new node list" on page 82
nlist s	"Show the contents of the current node list" on page 83
nlist sub *name*[*name*...]	"Delete node names from a list of nodes for mmpmon processing" on page 84

When specifying node names, keep these points in mind:

1. A node name of '.' (dot) indicates the current node.
2. A node name of '*' (asterisk) indicates all currently connected local cluster nodes.
3. The nodes named in the node list must belong to the local cluster. Nodes in remote clusters are not supported.
4. A node list can contain nodes that are currently down. When an inactive node comes up, **mmpmon** will attempt to gather data from it.
5. If a node list contains an incorrect or unrecognized node name, all other entries in the list are processed. Suitable messages are issued for an incorrect node name.
6. When **mmpmon** gathers responses from the nodes in a node list, the full response from one node is presented before the next node. Data is not interleaved. There is no guarantee of the order of node responses.
7. The node that issues the **mmpmon** command need not appear in the node list. The case of this node serving only as a collection point for data from other nodes is a valid configuration.

Add node names to a list of nodes for mmpmon processing

The **nlist add** (node list add) request is used to add node names to a list of nodes for **mmpmon** to collect their data. The node names are separated by blanks.

Table 8 describes the keywords for the **nlist add** response, in the order that they appear in the output. These keywords are used only when **mmpmon** is invoked with the **-p** flag.

Table 8. Keywords and values for the **mmpmon nlist add** *response*

Keyword	Description
_n_	IP address of the node processing the node list. This is the address by which GPFS knows the node.
_nn_	hostname that corresponds to the above IP address.
req	The action requested. In this case, the value is `add`.
_rc_	Indicates the status of the operation.

Table 8. Keywords and values for the **mmpmon nlist add** *response (continued)*

Keyword	Description
_t_	Current time of day in seconds (absolute seconds since Epoch (1970)).
_tu_	Microseconds part of the current time of day.
_c_	The number of nodes in the user-supplied list.
_ni_	Node name input. A user-supplied node name from the offered list of names.
_nx_	Node name translation. The preferred GPFS name for the node.
nxip	Node name translated IP address. The preferred GPFS IP address for the node.
did	The number of nodes names considered valid and processed by the requests.
nlc	The number of nodes in the node list now (after all processing).

If the **nlist add** request is issued when no node list exists, it is handled as if it were an **nlist new** request.

Example of mmpmon nlist add request

A two- node cluster has nodes **node1** (199.18.1.2), a non-quorum node, and **node2** (199.18.1.5), a quorum node. A remote cluster has node **node3** (199.18.1.8). The **mmpmon** command is run on **node1**.

Assume that **commandFile** contains this line:

```
nlist add n2 199.18.1.2
```

and this command is issued:

```
mmpmon -p -i commandFile
```

Note in this example that an alias name **n2** was used for **node2**, and an IP address was used for **node1**. Notice how the values for **_ni_** and **_nx_** differ in these cases.

The output is similar to this:

```
_nlist_ _n_ 199.18.1.2 _nn_ node1 _req_ add _rc_ 0 _t_ 1121955894 _tu_ 261881 _c_ 2
_nlist_ _n_ 199.18.1.2 _nn_ node1 _req_ add _rc_ 0 _t_ 1121955894 _tu_ 261881 _ni_ n2 _nx_
node2 _nxip_ 199.18.1.5
_nlist_ _n_ 199.18.1.2 _nn_ node1 _req_ add _rc_ 0 _t_ 1121955894 _tu_ 261881 _ni_
199.18.1.2 _nx_ node1 _nxip_ 199.18.1.2
_nlist_ _n_ 199.18.1.2 _nn_ node1 _req_ add _rc_ 0 _t_ 1121955894 _tu_ 261881 _did_ 2 _nlc_
2
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.2 name node1 nlist add
initial status 0
name count 2
timestamp 1121955879/468858
node name n2, OK (name used: node2, IP address 199.18.1.5)
node name 199.18.1.2, OK (name used: node1, IP address 199.18.1.2)
final status 0
node names processed 2
current node list count 2
```

The requests **nlist add** and **nlist sub** behave in a similar way and use the same keyword and response format.

These requests are rejected if issued while quorum has been lost.

Delete a node list

The **nlist del** (node list delete) request deletes a node list, if one exists. If no node list exists, the request succeeds and no error code is produced.

Table 9 describes the keywords for the **nlist del** response, in the order that they appear in the output. These keywords are used only when **mmpmon** is invoked with the **-p** flag.

Table 9. Keywords and values for the **mmpmon nlist del** *response*

Keyword	Description
_n_	IP address of the node responding. This is the address by which GPFS knows the node.
_nn_	hostname that corresponds to the above IP address.
req	The action requested. In this case, the value is `del`.
_rc_	Indicates the status of the operation.
_t_	Current time of day in seconds (absolute seconds since Epoch (1970)).
_tu_	Microseconds part of the current time of day.

Example of mmpmon nlist del request

Assume that **commandFile** contains this line:

```
nlist del
```

and this command is issued:

```
mmpmon -p -i commandFile
```

The output is similar to this:

```
_nlist_ _n_ 199.18.1.2 _nn_ node1 _req_ del _rc_ 0 _t_ 1121956817 _tu_ 46050
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.2 name node1 nlist del status OK timestamp 1121956908/396381
```

Create a new node list

The **nlist new** (node list new) request deletes the current node list if one exists, creates a new, empty node list, and then attempts to add the specified node names to the node list. The node names are separated by blanks.

Table 10 describes the keywords for the **nlist new** response, in the order that they appear in the output. These keywords are used only when **mmpmon** is invoked with the **-p** flag.

Table 10. Keywords and values for the **mmpmon nlist new** *response*

Keyword	Description
_n_	IP address of the node responding. This is the address by which GPFS knows the node.
_nn_	hostname that corresponds to the above IP address.
req	The action requested. In this case, the value is `new`.
_rc_	Indicates the status of the operation.
_t_	Current time of day in seconds (absolute seconds since Epoch (1970)).
_tu_	Microseconds part of the current time of day.

Show the contents of the current node list

The **nlist s** (node list show) request displays the current contents of the node list. If no node list exists, a count of zero is returned and no error is produced.

Table 11 describes the keywords for the **nlist s** response, in the order that they appear in the output. These keywords are used only when **mmpmon** is invoked with the **-p** flag.

Table 11. Keywords and values for the **mmpmon nlist s** *response*

Keyword	Description
_n_	IP address of the node processing the request. This is the address by which GPFS knows the node.
_nn_	hostname that corresponds to the above IP address.
req	The action requested. In this case, the value is `s`.
_rc_	Indicates the status of the operation.
_t_	Current time of day in seconds (absolute seconds since Epoch (1970)).
_tu_	Microseconds part of the current time of day.
_c_	Number of nodes in the node list.
mbr	GPFS preferred node name for the list member.
_ip_	GPFS preferred IP address for the list member.

Example of mmpmon nlist s request

Assume that **commandFile** contains this line:

```
nlist s
```

and this command is issued:

```
mmpmon -p -i commandFile
```

The output is similar to this:

```
_nlist_ _n_ 199.18.1.2 _nn_ node1 _req_ s _rc_ 0 _t_ 1121956950 _tu_ 863292 _c_ 2
_nlist_ _n_ 199.18.1.2 _nn_ node1 _req_ s _rc_ 0 _t_ 1121956950 _tu_ 863292 _mbr_ node1
_ip_ 199.18.1.2
_nlist_ _n_ 199.18.1.2 _nn_ node1 _req_ s _rc_ 0 _t_ 1121956950 _tu_ 863292 _mbr_
node2 _ip_ 199.18.1.5
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.2 name node1 nlist s
status 0
name count 2
timestamp 1121957505/165931
node name node1, IP address 199.18.1.2
node name node2, IP address 199.18.1.5
```

If there is no node list, the response looks like:

```
_nlist_ _n_ 199.18.1.2 _nn_ node1 _req_ s _rc_ 0 _t_ 1121957395 _tu_ 910440 _c_ 0
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.2 name node1 nlist s
status 0
name count 0
timestamp 1121957436/353352
the node list is empty
```

The **nlist s** request is rejected if issued while quorum has been lost. Only one response line is presented.

```
_failed_ _n_ 199.18.1.8 _nn_ node2 _rc_ 668 _t_ 1121957395 _tu_ 910440
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.8 name node2: failure status 668 timestamp 1121957395/910440
lost quorum
```

Delete node names from a list of nodes for mmpmon processing

The **nlist sub** (subtract a node from the node list) request removes a node from a list of node names. This keywords and responses are similar to the **nlist add** request. The **_req_** keyword (action requested) for **nlist sub** is sub.

Node list examples and error handling

The **nlist** facility can be used to obtain GPFS performance data from nodes other than the one on which the **mmpmon** command is invoked. This information is useful to see the flow of GPFS I/O from one node to another, and spot potential problems.

A successful fs_io_s request propagated to two nodes

In this example, an **fs_io_s** request is successfully propagated to two nodes. This command is issued:

```
mmpmon -p -i command_file
```

where **command_file** has this:

```
nlist new node1 node2
fs_io_s
```

The output is similar to this:

```
_fs_io_s_ _n_ 199.18.1.2 _nn_ node1 _rc_ 0 _t_ 1121974197 _tu_ 278619 _cl_
xxx.localdomain _fs_ gpfs2 _d_ 2 _br_ 0 _bw_ 0 _oc_ 0 _cc_ 0 _rdc_ 0 _wc_ 0
_dir_ 0 _iu_ 0
_fs_io_s_ _n_ 199.18.1.2 _nn_ node1 _rc_ 0 _t_ 1121974197 _tu_ 278619 _cl_
xxx.localdomain _fs_ gpfs1 _d_ 1 _br_ 0 _bw_ 0 _oc_ 0 _cc_ 0 _rdc_ 0 _wc_ 0
_dir_ 0 _iu_ 0
_fs_io_s_ _n_ 199.18.1.5 _nn_ node2 _rc_ 0 _t_ 1121974167 _tu_ 116443 _cl_
cl1.xxx.com _fs_ fs3 _d_ 3 _br_ 0 _bw_ 0 _oc_ 0 _cc_ 0 _rdc_ 0 _wc_ 0 _dir_ 0
_iu_ 3
_fs_io_s_ _n_ 199.18.1.5 _nn_ node2 _rc_ 0 _t_ 1121974167 _tu_ 116443 _cl_
cl1.xxx.comm _fs_ fs2 _d_ 2 _br_ 0 _bw_ 0 _oc_ 0 _cc_ 0 _rdc_ 0 _wc_ 0 _dir_ 0
_iu_ 0
_fs_io_s_ _n_ 199.18.1.5 _nn_ node2 _rc_ 0 _t_ 1121974167 _tu_ 116443 _cl_
xxx.localdomain _fs_ gpfs2 _d_ 2 _br_ 0 _bw_ 0 _oc_ 0 _cc_ 0 _rdc_ 0 _wc_ 0
_dir_ 0 _iu_ 0
```

The responses from a propagated request are the same as they would have been if issued on each node separately.

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.2 name node1 fs_io_s OK
cluster: xxx.localdomain
filesystem: gpfs2
disks: 2
timestamp: 1121974088/463102
bytes read: 0
bytes written: 0
opens: 0
closes: 0
reads: 0
```

```
writes: 0
readdir: 0
inode updates: 0

mmpmon node 199.18.1.2 name node1 fs_io_s OK
cluster: xxx.localdomain
filesystem: gpfs1
disks: 1
timestamp: 1121974088/463102
bytes read: 0
bytes written: 0
opens: 0
closes: 0
reads: 0
writes: 0
readdir: 0
inode updates: 0

mmpmon node 199.18.1.5 name node2 fs_io_s OK
cluster: cl1.xxx.com
filesystem: fs3
disks: 3
timestamp: 1121974058/321741
bytes read: 0
bytes written: 0
opens: 0
closes: 0
reads: 0
writes: 0
readdir: 0
inode updates: 2

mmpmon node 199.18.1.5 name node2 fs_io_s OK
cluster: cl1.xxx.com
filesystem: fs2
disks: 2
timestamp: 1121974058/321741
bytes read: 0
bytes written: 0
opens: 0
closes: 0
reads: 0
writes: 0
readdir: 0
inode updates: 0

mmpmon node 199.18.1.5 name node2 fs_io_s OK
cluster: xxx.localdomain
filesystem: gpfs2
disks: 2
timestamp: 1121974058/321741
bytes read: 0
bytes written: 0
opens: 0
closes: 0
reads: 0
writes: 0
readdir: 0
inode updates: 0
```

Failure on a node accessed by mmpmon

In this example, the same scenario is run on **node2**, but with a failure on **node1** (a non-quorum node) because **node1** was shutdown:

```
_failed_ _n_ 199.18.1.5 _nn_ node2 _fn_ 199.18.1.2 _fnn_ node1 _rc_ 233
_t_ 1121974459 _tu_ 602231
_fs_io_s_ _n_ 199.18.1.5 _nn_ node2 _rc_ 0 _t_ 1121974459 _tu_ 616867 _cl_
cl1.xxx.com _fs_ fs2 _d_ 2 _br_ 0 _bw_ 0 _oc_ 0 _cc_ 0 _rdc_ 0 _wc_ 0 _dir_ 0
_iu_ 0
_fs_io_s_ _n_ 199.18.1.5 _nn_ node2 _rc_ 0 _t_ 1121974459 _tu_ 616867 _cl_
cl1.xxx.com _fs_ fs3 _d_ 3 _br_ 0 _bw_ 0 _oc_ 0 _cc_ 0 _rdc_ 0 _wc_ 0 _dir_ 0
_iu_ 0
_fs_io_s_ _n_ 199.18.1.5 _nn_ node2 _rc_ 0 _t_ 1121974459 _tu_ 616867 _cl_
node1.localdomain _fs_ gpfs2 _d_ 2 _br_ 0 _bw_ 0 _oc_ 0 _cc_ 0 _rdc_ 0 _wc_ 0
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.5 name node2:
from node 199.18.1.2 from name node1: failure status 233 timestamp 1121974459/602231
node failed (or never started)
mmpmon node 199.18.1.5 name node2 fs_io_s OK
cluster: cl1.xxx.com
filesystem: fs2
disks: 2
timestamp: 1121974544/222514
bytes read: 0
bytes written: 0
opens: 0
closes: 0
reads: 0
writes: 0
readdir: 0
inode updates: 0

mmpmon node 199.18.1.5 name node2 fs_io_s OK
cluster: cl1.xxx.com
filesystem: fs3
disks: 3
timestamp: 1121974544/222514
bytes read: 0
bytes written: 0
opens: 0
closes: 0
reads: 0
writes: 0
readdir: 0
inode updates: 0

mmpmon node 199.18.1.5 name node2 fs_io_s OK
cluster: xxx.localdomain
filesystem: gpfs2
disks: 2
timestamp: 1121974544/222514
bytes read: 0
bytes written: 0
opens: 0
closes: 0
reads: 0
writes: 0
readdir: 0
inode updates: 0
```

Node shutdown and quorum loss

In this example, the quorum node (**node2**) is shutdown, causing quorum loss on **node1**. Running the same example on **node2**, the output is similar to:

```
_failed_ _n_ 199.18.1.2 _nn_ node1 _rc_ 668 _t_ 1121974459 _tu_ 616867
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.2 name node1: failure status 668 timestamp 1121974459/616867
lost quorum
```

In this scenario there can be a window where **node2** is down and **node1** has not yet lost quorum. When quorum loss occurs, the **mmpmon** command does not attempt to communicate with any nodes in the node list. The goal with failure handling is to accurately maintain the node list across node failures, so that when nodes come back up they again contribute to the aggregated responses.

Node list failure values

Table 12 describes the keywords and values produced by the **mmpmon** command on a node list failure:

Table 12. Keywords and values for the **mmpmon nlist** *failures*

Keyword	Description
_n_	IP address of the node processing the node list. This is the address by which GPFS knows the node.
_nn_	hostname that corresponds to the above IP address.
_fn_	IP address of the node that is no longer responding to **mmpmon** requests.
fnn	The name by which GPFS knows the node that is no longer responding to **mmpmon** requests
_rc_	Indicates the status of the operation. See “Return codes from mmpmon” on page 108.
_t_	Current time of day in seconds (absolute seconds since Epoch (1970)).
_tu_	Microseconds part of the current time of day.

Understanding the request histogram facility

The **mmpmon** requests whose name starts with **rhist** control the request histogram facility. This facility tallies I/O operations using a set of counters. Counters for reads and writes are kept separately. They are categorized according to a pattern that may be customized by the user. A default pattern is also provided. The **size range** and **latency range** input parameters to the **rhist nr** request are used to define the pattern.

The first time that you run the **rhist** requests, assess if there is a noticeable performance degradation. Collecting histogram data may cause performance degradation. This is possible once the histogram facility is enabled, but will probably not be noticed while the commands themselves are running. It is more of a long term issue as the GPFS daemon runs with histograms enabled.

The histogram lock is used to prevent two **rhist** requests from being processed simultaneously. If an **rhist** request fails with an **_rc_** of 16, the lock is in use. Reissue the request.

The histogram data survives file system mounts and unmounts. In order to reset this data, use the **rhist reset** request.

Specifying the size ranges for I/O histograms

The size ranges are used to categorize the I/O according to the size, in bytes, of the I/O operation. The size ranges are specified using a string of positive integers separated by semicolons (;). No white space is allowed within the size range operand. Each number represents the upper bound, in bytes, of the I/O request size for that range. The numbers must be monotonically increasing. Each number may be optionally followed by the letters K or k to denote multiplication by 1024, or by the letters M or m to denote multiplication by 1048576 (1024*1024).

For example, the size range operand:

```
512;1m;4m
```

represents these four size ranges

```
0        to     512 bytes
513      to 1048576 bytes
1048577  to 4194304 bytes
4194305 and greater bytes
```

In this example, a read of size 3 MB would fall in the third size range, a write of size 20 MB would fall in the fourth size range.

A size range operand of = (equal sign) indicates that the current size range is not to be changed. A size range operand of * (asterisk) indicates that the current size range is to be changed to the default size range. A maximum of 15 numbers may be specified, which produces 16 total size ranges.

The default request size ranges are:

```
0        to      255 bytes
256      to      511 bytes
512      to     1023 bytes
1024     to     2047 bytes
2048     to     4095 bytes
4096     to     8191 bytes
8192     to    16383 bytes
16384    to    32767 bytes
32768    to    65535 bytes
65536    to   131071 bytes
131072   to   262143 bytes
262144   to   524287 bytes
524288   to  1048575 bytes
1048576  to  2097151 bytes
2097152  to  4194303 bytes
4194304  and greater bytes
```

The last size range collects all request sizes greater than or equal to 4 MB. The request size ranges can be changed by using the **rhist nr** request.

Specifying the latency ranges for I/O

The latency ranges are used to categorize the I/O according to the latency time, in milliseconds, of the I/O operation. A full set of latency ranges are produced for each size range. The latency ranges are the same for each size range.

The latency ranges are changed using a string of positive decimal numbers separated by semicolons (;). No white space is allowed within the latency range operand. Each number represents the upper bound of the I/O latency time (in milliseconds) for that range. The numbers must be monotonically increasing. If decimal places are present, they are truncated to tenths.

For example, the latency range operand:

```
1.3;4.59;10
```

represents these four latency ranges:

```
 0.0   to   1.3   milliseconds
 1.4   to   4.5   milliseconds
 4.6   to  10.0   milliseconds
10.1  and greater milliseconds
```

In this example, a read that completes in 0.85 milliseconds falls into the first latency range. A write that completes in 4.56 milliseconds falls into the second latency range, due to the truncation.

A latency range operand of = (equal sign) indicates that the current latency range is not to be changed. A latency range operand of * (asterisk) indicates that the current latency range is to be changed to the

default latency range. If the latency range operand is missing, * (asterisk) is assumed. A maximum of 15 numbers may be specified, which produces 16 total latency ranges.

The latency times are in milliseconds. The default latency ranges are:

```
0.0     to    1.0   milliseconds
1.1     to   10.0   milliseconds
10.1    to   30.0   milliseconds
30.1    to  100.0   milliseconds
100.1   to  200.0   milliseconds
200.1   to  400.0   milliseconds
400.1   to  800.0   milliseconds
800.1   to 1000.0   milliseconds
1000.1 and greater  milliseconds
```

The last latency range collects all latencies greater than or equal to 1000.1 milliseconds. The latency ranges can be changed by using the **rhist nr** request.

Changing the request histogram facility request size and latency ranges

The **rhist nr** (new range) request allows the user to change the size and latency ranges used in the request histogram facility. The use of **rhist nr** implies an **rhist reset**. Counters for read and write operations are recorded separately. If there are no mounted file systems at the time **rhist nr** is issued, the request still runs. The size range operand appears first, followed by a blank, and then the latency range operand.

Table 13 describes the keywords for the **rhist nr** response, in the order that they appear in the output. These keywords are used only when **mmpmon** is invoked with the **-p** flag.

Table 13. Keywords and values for the **mmpmon rhist nr** *response*

Keyword	Description
_n_	IP address of the node responding. This is the address by which GPFS knows the node.
_nn_	hostname that corresponds to the above IP address.
req	The action requested. In this case, the value is `nr`.
_rc_	Indicates the status of the operation.
_t_	Current time of day in seconds (absolute seconds since Epoch (1970)).
_tu_	Microseconds part of the current time of day.

An _**rc**_ value of 16 indicates that the histogram operations lock is busy. Retry the request.

Processing of rhist nr

Processing of **rhist nr** is as follows:

1. The size range and latency range operands are parsed and checked for validity. If they are not valid, an error is returned and processing terminates.
2. The histogram facility is disabled.
3. The new ranges are created, by defining the following histogram counters:
   a. Two sets, one for read and one for write.
   b. Within each set, one category for each size range.
   c. Within each size range category, one counter for each latency range.

For example, if the user specifies 11 numbers for the size range operand and 2 numbers for the latency range operand, this produces 12 size ranges, each having 3 latency ranges, because there is one additional range for the top endpoint. The total number of counters is 72: 36 read counters and 36 write counters.

4. The new ranges are made current.
5. The old ranges are discarded. Any accumulated histogram data is lost.

The histogram facility must be explicitly enabled again using **rhist on** to begin collecting histogram data using the new ranges.

The **mmpmon** command does not have the ability to collect data only for read operations, or only for write operations. The **mmpmon** command does not have the ability to specify size or latency ranges that have different values for read and write operations. The **mmpmon** command does not have the ability to specify latency ranges that are unique to a given size range.

Example of mmpmon rhist nr request

Assume that **commandFile** contains this line:

```
rhist nr 512;1m;4m 1.3;4.5;10
```

and this command is issued:

```
mmpmon -p -i commandFile
```

The output is similar to this:

```
_rhist_ _n_ 199.18.2.5 _nn_ node1 _req_ nr 512;1m;4m 1.3;4.5;10 _rc_ 0 _t_ 1078929833 _tu_ 765083
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.8 name node1 rhist nr 512;1m;4m 1.3;4.5;10 OK
```

In this case, **mmpmon** has been instructed to keep a total of 32 counters. There are 16 for read and 16 for write. For the reads, there are four size ranges, each of which has four latency ranges. The same is true for the writes. They are as follows:

```
size range            0   to   512    bytes
  latency range       0.0   to   1.3    milliseconds
  latency range       1.4   to   4.5    milliseconds
  latency range       4.6   to  10.0    milliseconds
  latency range       10.1 and greater  milliseconds
size range          513   to  1048576 bytes
  latency range       0.0   to   1.3    milliseconds
  latency range       1.4   to   4.5    milliseconds
  latency range       4.6   to  10.0    milliseconds
  latency range       10.1 and greater  milliseconds
size range       1048577  to  4194304 bytes
  latency range       0.0   to   1.3    milliseconds
  latency range       1.4   to   4.5    milliseconds
  latency range       4.6   to  10.0    milliseconds
  latency range       10.1 and greater  milliseconds
size range       4194305 and greater bytes
  latency range       0.0   to   1.3    milliseconds
  latency range       1.4   to   4.5    milliseconds
  latency range       4.6   to  10.0    milliseconds
  latency range       10.1 and greater  milliseconds
```

In this example, a read of size 15 MB that completes in 17.8 milliseconds would fall in the last latency range listed here. When this read completes, the counter for the last latency range will be increased by one.

An **_rc_** value of 16 indicates that the histogram operations lock is busy. Retry the request.

An example of an unsuccessful response is:

```
_rhist_ _n_ 199.18.2.5 _nn_ node1 _req_ nr 512;1m;4m 1;4;8;2 _rc_ 22 _t_ 1078929596 _tu_ 161683
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.8 name node1 rhist nr 512;1m;4m 1;4;8;2 status 22 range error
```

In this case, the last value in the latency range, 2, is out of numerical order.

Note that the request **rhist nr = =** does not make any changes. It is ignored.

Disabling the request histogram facility

The **rhist off** request disables the request histogram facility. The data objects remain persistent, and the data they contain is not disturbed. This data is not updated again until **rhist on** is issued. **rhist off** may be combined with **rhist on** as often as desired. If there are no mounted file systems at the time **rhist off** is issued, the facility is still disabled. The response is a single string.

Table 14 describes the keywords for the **rhist off** response, in the order that they appear in the output. These keywords are used only when **mmpmon** is invoked with the **-p** flag.

Table 14. Keywords and values for the **mmpmon rhist off** *response*

Keyword	Description
_n_	IP address of the node responding. This is the address by which GPFS knows the node.
_nn_	hostname that corresponds to the above IP address.
req	The action requested. In this case, the value is `off`.
_rc_	Indicates the status of the operation.
_t_	Current time of day in seconds (absolute seconds since Epoch (1970)).
_tu_	Microseconds part of the current time of day.

An **_rc_** value of 16 indicates that the histogram operations lock is busy. Retry the request.

Example of mmpmon rhist off request

Assume that **commandFile** contains this line:

```
rhist off
```

and this command is issued:

```
mmpmon -p -i commandFile
```

The output is similar to this:

```
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ off _rc_ 0 _t_ 1066938820 _tu_ 5755
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.8 name node1 rhist off OK
```

An **_rc_** value of 16 indicates that the histogram operations lock is busy. Retry the request.

```
mmpmon node 199.18.1.8 name node1 rhist off status 16
lock is busy
```

Enabling the request histogram facility

The **rhist on** request enables the request histogram facility. When invoked the first time, this request creates the necessary data objects to support histogram data gathering. This request may be combined with **rhist off** (or another **rhist on**) as often as desired. If there are no mounted file systems at the time **rhist on** is issued, the facility is still enabled. The response is a single string.

Table 15 describes the keywords for the **rhist on** response, in the order that they appear in the output. These keywords are used only when **mmpmon** is invoked with the **-p** flag.

Table 15. Keywords and values for the **mmpmon rhist on** *response*

Keyword	Description
_n_	IP address of the node responding. This is the address by which GPFS knows the node.
_nn_	hostname that corresponds to the above IP address.
req	The action requested. In this case, the value is `on`.
_rc_	Indicates the status of the operation.
_t_	Current time of day in seconds (absolute seconds since Epoch (1970)).
_tu_	Microseconds part of the current time of day.

An _**rc**_ value of 16 indicates that the histogram operations lock is busy. Retry the request.

Example of mmpmon rhist on request

Assume that **commandFile** contains this line:

```
rhist on
```

and this command is issued:

```
mmpmon -p -i commandFile
```

The output is similar to this:

```
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ on _rc_ 0 _t_ 1066936484 _tu_ 179346
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.8 name node1 rhist on OK
```

An _**rc**_ value of 16 indicates that the histogram operations lock is busy. Retry the request.

```
mmpmon node 199.18.1.8 name node1 rhist on status 16
lock is busy
```

Displaying the request histogram facility pattern

The **rhist p** request returns the entire enumeration of the request size and latency ranges. The facility must be enabled for a pattern to be returned. If there are no mounted file systems at the time this request is issued, the request still runs and returns data. The pattern is displayed for both read and write.

Table 16 describes the keywords for the **rhist p** response, in the order that they appear in the output. These keywords are used only when **mmpmon** is invoked with the **-p** flag.

Table 16. Keywords and values for the **mmpmon rhist p** *response*

Keyword	Description
_n_	IP address of the node responding. This is the address by which GPFS knows the node.

Table 16. Keywords and values for the **mmpmon rhist p** *response (continued)*

Keyword	Description
_nn_	hostname that corresponds to the above IP address.
req	The action requested. In this case, the value is `p`.
_rc_	Indicates the status of the operation.
_t_	Current time of day in seconds (absolute seconds since Epoch (1970)).
_tu_	Microseconds part of the current time of day.
_k_	The kind, **r** or **w**, (read or write) depending on what the statistics are for.
_R_	Request size range, minimum and maximum number of bytes.
_L_	Latency range, minimum and maximum, in milliseconds.

The request size ranges are in bytes. The zero value used for the upper limit of the last size range means 'and above'. The request size ranges can be changed by using the **rhist nr** request.

The latency times are in milliseconds The zero value used for the upper limit of the last latency range means 'and above'. The latency ranges can be changed by using the **rhist nr** request.

The **rhist p** request allows an application to query for the entire latency pattern. The application can then configure itself accordingly. Since latency statistics are reported only for ranges with nonzero counts, the statistics responses may be sparse. By querying for the pattern, an application can be certain to learn the complete histogram set. The user may have changed the pattern using the **rhist nr** request. For this reason, an application should query for the pattern and analyze it before requesting statistics.

If the facility has never been enabled, the _**rc**_ field will be nonzero. An _**rc**_ value of 16 indicates that the histogram operations lock is busy. Retry the request.

If the facility has been previously enabled, the **rhist p** request will still display the pattern even if **rhist off** is currently in effect.

If there are no mounted file systems at the time **rhist p** is issued, the pattern is still displayed.

Example of mmpmon rhist p request

Assume that **commandFile** contains this line:

```
rhist p
```

and this command is issued:

```
mmpmon -p -i commandFile
```

The response contains all the latency ranges inside each of the request ranges. The data are separate for read and write:

```
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ p _rc_ 0 _t_ 1066939007 _tu_ 386241 _k_ r
... data for reads ...
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ p _rc_ 0 _t_ 1066939007 _tu_ 386241 _k_ w
... data for writes ...
_end_
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.8 name node1 rhist p OK read
... data for reads ...
mmpmon node 199.188.1.8 name node1 rhist p OK write
... data for writes ...
```

Here is an example of data for reads:

```
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ p _rc_ 0 _t_ 1066939007 _tu_ 386241 _k_ r
_R_          0            255
_L_        0.0            1.0
_L_        1.1           10.0
_L_       10.1           30.0
_L_       30.1          100.0
_L_      100.1          200.0
_L_      200.1          400.0
_L_      400.1          800.0
_L_      800.1         1000.0
_L_     1000.1              0
_R_        256            511
_L_        0.0            1.0
_L_        1.1           10.0
_L_       10.1           30.0
_L_       30.1          100.0
_L_      100.1          200.0
_L_      200.1          400.0
_L_      400.1          800.0
_L_      800.1         1000.0
_L_     1000.1              0
_R_        512           1023
_L_        0.0            1.0
_L_        1.1           10.0
_L_       10.1           30.0
_L_       30.1          100.0
_L_      100.1          200.0
_L_      200.1          400.0
_L_      400.1          800.0
_L_      800.1         1000.0
_L_     1000.1              0
...
_R_    4194304              0
_L_        0.0            1.0
_L_        1.1           10.0
_L_       10.1           30.0
_L_       30.1          100.0
_L_      100.1          200.0
_L_      200.1          400.0
_L_      400.1          800.0
_L_      800.1         1000.0
_L_     1000.1              0
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.8 name node1 rhist p OK read
size range          0 to          255
  latency range    0.0 to          1.0
  latency range    1.1 to         10.0
  latency range   10.1 to         30.0
  latency range   30.1 to        100.0
  latency range  100.1 to        200.0
  latency range  200.1 to        400.0
  latency range  400.1 to        800.0
  latency range  800.1 to       1000.0
  latency range 1000.1 to            0
size range        256 to          511
  latency range    0.0 to          1.0
  latency range    1.1 to         10.0
  latency range   10.1 to         30.0
  latency range   30.1 to        100.0
  latency range  100.1 to        200.0
  latency range  200.1 to        400.0
  latency range  400.1 to        800.0
  latency range  800.1 to       1000.0
  latency range 1000.1 to            0
```

```
size range       512 to         1023
  latency range    0.0 to          1.0
  latency range    1.1 to         10.0
  latency range   10.1 to         30.0
  latency range   30.1 to        100.0
  latency range  100.1 to        200.0
  latency range  200.1 to        400.0
  latency range  400.1 to        800.0
  latency range  800.1 to       1000.0
  latency range 1000.1 to            0
...
size range   4194304 to            0
  latency range    0.0 to          1.0
  latency range    1.1 to         10.0
  latency range   10.1 to         30.0
  latency range   30.1 to        100.0
  latency range  100.1 to        200.0
  latency range  200.1 to        400.0
  latency range  400.1 to        800.0
  latency range  800.1 to       1000.0
  latency range 1000.1 to            0
```

If the facility has never been enabled, the _**rc**_ field will be nonzero.

```
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ p _rc_ 1 _t_ 1066939007 _tu_ 386241
```

If the **-p** flag is not specified, the output is similar to this:

```
mmpmon node 199.18.1.8 name node1 rhist p status 1
not yet enabled
```

Resetting the request histogram facility data to zero

The **rhist reset** request resets the histogram statistics. Table 17 describes the keywords for the **rhist reset** response, in the order that they appear in the output. These keywords are used only when **mmpmon** is invoked with the **-p** flag. The response is a single string.

Table 17. Keywords and values for the **mmpmon rhist reset** *response*

Keyword	Description
_n_	IP address of the node responding. This is the address by which GPFS knows the node.
_nn_	hostname that corresponds to the above IP address.
req	The action requested. In this case, the value is `reset`.
_rc_	Indicates the status of the operation.
_t_	Current time of day in seconds (absolute seconds since Epoch (1970)).
_tu_	Microseconds part of the current time of day.

If the facility has been previously enabled, the reset request will still reset the statistics even if **rhist off** is currently in effect. If there are no mounted file systems at the time **rhist reset** is issued, the statistics are still reset.

An _**rc**_ value of 16 indicates that the histogram operations lock is busy. Retry the request.

Example of mmpmon rhist reset request

Assume that **commandFile** contains this line:

```
rhist reset
```

and this command is issued:

```
mmpmon -p -i commandFile
```

The output is similar to this:

```
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ reset _rc_ 0 _t_ 1066939007 _tu_ 386241
```

If the **-p** flag is not specified, the output is similar to:

```
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ reset _rc_ 0 _t_ 1066939007 _tu_ 386241
```

If the facility has never been enabled, the _**rc**_ value will be nonzero:

```
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ reset _rc_ 1 _t_ 1066939143 _tu_ 148443
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.8 name node1 rhist reset status 1
not yet enabled
```

Displaying the request histogram facility statistics values

The **rhist s** request returns the current values for all latency ranges which have a nonzero count.

Table 18 describes the keywords for the **rhist s** response, in the order that they appear in the output. These keywords are used only when **mmpmon** is invoked with the **-p** flag.

Table 18. Keywords and values for the **mmpmon rhist s** *response*

Keyword	Description
_n_	IP address of the node responding. This is the address by which GPFS knows the node.
_nn_	hostname that corresponds to the above IP address.
req	The action requested. In this case, the value is `s`.
_rc_	Indicates the status of the operation.
_t_	Current time of day in seconds (absolute seconds since Epoch (1970)).
_tu_	Microseconds part of the current time of day.
_k_	The kind, **r** or **w**, (read or write) depending on what the statistics are for.
_R_	Request size range, minimum and maximum number of bytes.
_NR_	Number of requests that fell in this size range.
_L_	Latency range, minimum and maximum, in milliseconds.
_NL_	Number of requests that fell in this latency range. The sum of all _**NL**_ values for a request size range equals the _**NR**_ value for that size range.

If the facility has been previously enabled, the **rhist s** request will still display the statistics even if **rhist off** is currently in effect. This allows turning the histogram statistics on and off between known points and reading them later. If there are no mounted file systems at the time **rhist s** is issued, the statistics are still displayed.

An _**rc**_ value of 16 indicates that the histogram operations lock is busy. Retry the request.

Example of mmpmon rhist s request

Assume that **commandFile** contains this line:

```
rhist s
```

and this command is issued:

```
mmpmon -p -i commandFile
```

The output is similar to this:

```
_rhist_ _n_ 199.18.2.5 _nn_ node1 _req_ s _rc_ 0 _t_ 1066939007 _tu_ 386241 _k_ r
_R_       65536      131071 _NR_      32640
_L_         0.0         1.0 _NL_      25684
_L_         1.1        10.0 _NL_       4826
_L_        10.1        30.0 _NL_       1666
_L_        30.1       100.0 _NL_        464
_R_      262144      524287 _NR_       8160
_L_         0.0         1.0 _NL_       5218
_L_         1.1        10.0 _NL_        871
_L_        10.1        30.0 _NL_       1863
_L_        30.1       100.0 _NL_        208
_R_     1048576     2097151 _NR_       2040
_L_         1.1        10.0 _NL_        558
_L_        10.1        30.0 _NL_        809
_L_        30.1       100.0 _NL_        673
_rhist_ _n_ 199.18.2.5 _nn_ node1 _req_ s _rc_ 0 _t_ 1066939007 _tu_ 386241 _k_ w
_R_      131072      262143 _NR_      12240
_L_         0.0         1.0 _NL_      10022
_L_         1.1        10.0 _NL_       1227
_L_        10.1        30.0 _NL_        783
_L_        30.1       100.0 _NL_        208
_R_      262144      524287 _NR_       6120
_L_         0.0         1.0 _NL_       4419
_L_         1.1        10.0 _NL_        791
_L_        10.1        30.0 _NL_        733
_L_        30.1       100.0 _NL_        177
_R_      524288     1048575 _NR_       3060
_L_         0.0         1.0 _NL_       1589
_L_         1.1        10.0 _NL_        581
_L_        10.1        30.0 _NL_        664
_L_        30.1       100.0 _NL_        226
_R_     2097152     4194303 _NR_        762
_L_         1.1         2.0 _NL_        203
_L_        10.1        30.0 _NL_        393
_L_        30.1       100.0 _NL_        166
_end_
```

This small example shows that the reports for read and write may not present the same number of ranges or even the same ranges. Only those ranges with nonzero counters are represented in the response. This is true for both the request size ranges and the latency ranges within each request size range.

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.2.5 name node1 rhist s OK timestamp 1066933849/93804 read
size range            65536 to    131071 count      32640
  latency range         0.0 to       1.0 count      25684
  latency range         1.1 to      10.0 count       4826
  latency range        10.1 to      30.0 count       1666
  latency range        30.1 to     100.0 count        464
size range           262144 to    524287 count       8160
  latency range         0.0 to       1.0 count       5218
  latency range         1.1 to      10.0 count        871
  latency range        10.1 to      30.0 count       1863
  latency range        30.1 to     100.0 count        208
size range          1048576 to    2097151 count      2040
  latency range         1.1 to       10.0 count       558
  latency range        10.1 to       30.0 count       809
  latency range        30.1 to      100.0 count       673
mmpmon node 199.18.2.5 name node1 rhist s OK timestamp 1066933849/93968 write
size range           131072 to     262143 count     12240
  latency range         0.0 to        1.0 count     10022
  latency range         1.1 to       10.0 count      1227
  latency range        10.1 to       30.0 count       783
  latency range        30.1 to      100.0 count       208
size range           262144 to     524287 count      6120
```

```
  latency range         0.0 to        1.0 count      4419
  latency range         1.1 to       10.0 count       791
  latency range        10.1 to       30.0 count       733
  latency range        30.1 to      100.0 count       177
size range           524288 to    1048575 count      3060
  latency range         0.0 to        1.0 count      1589
  latency range         1.1 to       10.0 count       581
  latency range        10.1 to       30.0 count       664
  latency range        30.1 to      100.0 count       226
size range          2097152 to     4194303 count      762
  latency range         1.1 to         2.0 count      203
  latency range        10.1 to        30.0 count      393
  latency range        30.1 to       100.0 count     166
```

If the facility has never been enabled, the _**rc**_ value will be nonzero:

```
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ reset _rc_ 1 _t_ 1066939143 _tu_ 148443
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.8 name node1 rhist reset status 1
not yet enabled
```

An _**rc**_ value of 16 indicates that the histogram operations lock is busy. Retry the request.

Displaying mmpmon version

The **ver** request returns a string containing version information. Table 19 describes the keywords for the **ver** (version) response, in the order that they appear in the output. These keywords are used only when **mmpmon** is invoked with the **-p** flag.

Table 19. Keywords and values for the **mmpmon ver** *response*

Keyword	Description
_n_	IP address of the node responding. This is the address by which GPFS knows the node.
_nn_	hostname that corresponds to the above IP address.
_v_	The version of **mmpmon**.
_lv_	The level of **mmpmon**.
_vt_	The fix level variant of **mmpmon**.

Example of mmpmon ver request

Assume that **commandFile** contains this line:

```
ver
```

and this command is issued:

```
mmpmon -p -i commandFile
```

The output is similar to this:

```
_ver_ _n_ 199.18.1.8 _nn_ node1 _v_ 2 _lv_ 3 _vt_ 0
```

If the **-p** flag is not specified, the output is similar to:

```
mmpmon node 199.18.1.8 name node1 version 3.1.0
```

Example mmpmon scenarios and how to analyze and interpret their results

This topic is an illustration of how **mmpmon** is used to analyze I/O data and draw conclusions based on it.

The **fs_io_s** and **io_s** requests are used to determine a number of GPFS I/O parameters and their implication for overall performance. The **rhist** requests are used to produce histogram data about I/O sizes and latency times for I/O requests. The request *source* and prefix directive *once* allow the user of **mmpmon** to more finely tune its operation.

fs_io_s and io_s output - how to aggregate and analyze the results

The output from the **fs_io_s** and **io_s** requests can be used to determine:

1. The I/O service rate of a node, from the application point of view. The **io_s** request presents this as a sum for the entire node, while **fs_io_s** presents the data per file system. A rate can be approximated by taking the **_br_** (bytes read) or **_bw_** (bytes written) values from two successive invocations of **fs_io_s** (or **io_s_**) and dividing by the difference of the sums of the individual **_t_** and **_tu_** values (seconds and microseconds).

   This must be done for a number of samples, with a reasonably small time between samples, in order to get a rate which is reasonably accurate. Since we are sampling the information at a given interval, inaccuracy can exist if the I/O load is not smooth over the sampling time.

   For example, here is a set of samples taken approximately one second apart, when it was known that continuous I/O activity was occurring:

```
_fs_io_s_ _n_ 199.18.1.3 _nn_ node1 _rc_ 0 _t_ 1095862476 _tu_ 634939 _cl_ cluster1.xxx.com
_fs_ gpfs1m _d_ 3 _br_ 0 _bw_ 3737124864 _oc_ 4 _cc_ 3 _rdc_ 0 _wc_ 3570 _dir_ 0 _iu_ 5
_fs_io_s_ _n_ 199.18.1.3 _nn_ node1 _rc_ 0 _t_ 1095862477 _tu_ 645988 _cl_ cluster1.xxx.com
_fs_ gpfs1m _d_ 3 _br_ 0 _bw_ 3869245440 _oc_ 4 _cc_ 3 _rdc_ 0 _wc_ 3696 _dir_ 0 _iu_ 5
_fs_io_s_ _n_ 199.18.1.3 _nn_ node1 _rc_ 0 _t_ 1095862478 _tu_ 647477 _cl_ cluster1.xxx.com
_fs_ gpfs1m _d_ 3 _br_ 0 _bw_ 4120903680 _oc_ 4 _cc_ 3 _rdc_ 0 _wc_ 3936 _dir_ 0 _iu_ 5
_fs_io_s_ _n_ 199.18.1.3 _nn_ node1 _rc_ 0 _t_ 1095862479 _tu_ 649363 _cl_ cluster1.xxx.com
_fs_ gpfs1m _d_ 3 _br_ 0 _bw_ 4309647360 _oc_ 4 _cc_ 3 _rdc_ 0 _wc_ 4116 _dir_ 0 _iu_ 5
_fs_io_s_ _n_ 199.18.1.3 _nn_ node1 _rc_ 0 _t_ 1095862480 _tu_ 650795 _cl_ cluster1.xxx.com
_fs_ gpfs1m _d_ 3 _br_ 0 _bw_ 4542431232 _oc_ 4 _cc_ 3 _rdc_ 0 _wc_ 4338 _dir_ 0 _iu_ 5
_fs_io_s_ _n_ 199.18.1.3 _nn_ node1 _rc_ 0 _t_ 1095862481 _tu_ 652515 _cl_ cluster1.ibm.com
_fs_ gpfs1m _d_ 3 _br_ 0 _bw_ 4743757824 _oc_ 4 _cc_ 3 _rdc_ 0 _wc_ 4530 _dir_ 0 _iu_ 5
_fs_io_s_ _n_ 199.18.1.3 _nn_ node1 _rc_ 0 _t_ 1095862482 _tu_ 654025 _cl_ cluster1.xxx.com
_fs_ gpfs1m _d_ 3 _br_ 0 _bw_ 4963958784 _oc_ 4 _cc_ 3 _rdc_ 0 _wc_ 4740 _dir_ 0 _iu_ 5
_fs_io_s_ _n_ 199.18.1.3 _nn_ node1 _rc_ 0 _t_ 1095862483 _tu_ 655782 _cl_ cluster1.xxx.com
_fs_ gpfs1m _d_ 3 _br_ 0 _bw_ 5177868288 _oc_ 4 _cc_ 3 _rdc_ 0 _wc_ 4944 _dir_ 0 _iu_ 5
_fs_io_s_ _n_ 199.18.1.3 _nn_ node1 _rc_ 0 _t_ 1095862484 _tu_ 657523 _cl_ cluster1.xxx.com
_fs_ gpfs1m _d_ 3 _br_ 0 _bw_ 5391777792 _oc_ 4 _cc_ 3 _rdc_ 0 _wc_ 5148 _dir_ 0 _iu_ 5
_fs_io_s_ _n_ 199.18.1.3 _nn_ node1 _rc_ 0 _t_ 1095862485 _tu_ 665909 _cl_ cluster1.xxx.com
_fs_ gpfs1m _d_ 3 _br_ 0 _bw_ 5599395840 _oc_ 4 _cc_ 3 _rdc_ 0 _wc_ 5346 _dir_ 0 _iu_ 5
```

   This simple **awk** script performs a basic rate calculation:

```
BEGIN {
  count=0;
  prior_t=0;
  prior_tu=0;
  prior_br=0;
  prior_bw=0;
}

{
  count++;

  t = $9;
  tu = $11;
```

```
  br = $19;
  bw = $21;

  if(count > 1)
  {
    delta_t = t-prior_t;
    delta_tu = tu-prior_tu;
    delta_br = br-prior_br;
    delta_bw = bw-prior_bw;
    dt = delta_t + (delta_tu / 1000000.0);
    if(dt > 0) {
      rrate = (delta_br / dt) / 1000000.0;
      wrate = (delta_bw / dt) / 1000000.0;
      printf("
    }
  }

  prior_t=t;
  prior_tu=tu;
  prior_br=br;
  prior_bw=bw;
}
```

The calculated service rates for each adjacent pair of samples is:

```
  0.0 MB/sec read    130.7 MB/sec write
  0.0 MB/sec read    251.3 MB/sec write
  0.0 MB/sec read    188.4 MB/sec write
  0.0 MB/sec read    232.5 MB/sec write
  0.0 MB/sec read    201.0 MB/sec write
  0.0 MB/sec read    219.9 MB/sec write
  0.0 MB/sec read    213.5 MB/sec write
  0.0 MB/sec read    213.5 MB/sec write
  0.0 MB/sec read    205.9 MB/sec write
```

Since these are discrete samples, there can be variations in the individual results. For example, there may be other activity on the node or interconnection fabric. I/O size, file system block size, and buffering also affect results. There can be many reasons why adjacent values differ. This must be taken into account when building analysis tools that read **mmpmon** output and interpreting results.

For example, suppose a file is read for the first time and gives results like this.

```
  0.0 MB/sec read      0.0 MB/sec write
  0.0 MB/sec read      0.0 MB/sec write
 92.1 MB/sec read      0.0 MB/sec write
 89.0 MB/sec read      0.0 MB/sec write
 92.1 MB/sec read      0.0 MB/sec write
 90.0 MB/sec read      0.0 MB/sec write
 96.3 MB/sec read      0.0 MB/sec write
  0.0 MB/sec read      0.0 MB/sec write
  0.0 MB/sec read      0.0 MB/sec write
```

If most or all of the file remains in the GPFS cache, the second read may give quite different rates:

```
  0.0 MB/sec read      0.0 MB/sec write
  0.0 MB/sec read      0.0 MB/sec write
235.5 MB/sec read      0.0 MB/sec write
287.8 MB/sec read      0.0 MB/sec write
  0.0 MB/sec read      0.0 MB/sec write
  0.0 MB/sec read      0.0 MB/sec write
```

Considerations such as these need to be taken into account when looking at application I/O service rates calculated from sampling **mmpmon** data.

2. Usage patterns, by sampling at set times of the day (perhaps every half hour) and noticing when the largest changes in I/O volume occur. This does not necessarily give a rate (since there are too few samples) but it can be used to detect peak usage periods.
3. If some nodes service significantly more I/O volume than others over a given time span.

4. When a parallel application is split across several nodes, and is the only significant activity in the nodes, how well the I/O activity of the application is distributed.
5. The total I/O demand that applications are placing on the cluster. This is done by obtaining results from **fs_io_s** and **io_s** in aggregate for all nodes in a cluster.
6. The rate data may appear to be erratic. Consider this example:

```
 0.0 MB/sec read    0.0 MB/sec write
 6.1 MB/sec read    0.0 MB/sec write
92.1 MB/sec read    0.0 MB/sec write
89.0 MB/sec read    0.0 MB/sec write
12.6 MB/sec read    0.0 MB/sec write
 0.0 MB/sec read    0.0 MB/sec write
 0.0 MB/sec read    0.0 MB/sec write
 8.9 MB/sec read    0.0 MB/sec write
92.1 MB/sec read    0.0 MB/sec write
90.0 MB/sec read    0.0 MB/sec write
96.3 MB/sec read    0.0 MB/sec write
 4.8 MB/sec read    0.0 MB/sec write
 0.0 MB/sec read    0.0 MB/sec write
```

   The low rates which appear before and after each group of higher rates can be due to the I/O requests occurring late (in the leading sampling period) and ending early (in the trailing sampling period.) This gives an apparently low rate for those sampling periods.

   The zero rates in the middle of the example could be caused by reasons such as no I/O requests reaching GPFS during that time period (the application issued none, or requests were satisfied by buffered data at a layer above GPFS), the node becoming busy with other work (causing the application to be undispatched), or other reasons.

Request histogram (rhist) output - how to aggregate and analyze the results

The output from the **rhist** requests can be used to determine:

1. The number of I/O requests in a given size range. The sizes may vary based on operating system, explicit application buffering, and other considerations. This information can be used to help determine how well an application or set of applications is buffering its I/O. For example, if are there many very small or many very large I/O transactions. A large number of overly small or overly large I/O requests may not perform as well as an equivalent number of requests whose size is tuned to the file system or operating system parameters.
2. The number of I/O requests in a size range that have a given latency time. Many factors can affect the latency time, including but not limited to: system load, interconnection fabric load, file system block size, disk block size, disk hardware characteristics, and the operating system on which the I/O request is issued.

Using request *source* and prefix directive *once*

The **source** request causes **mmpmon** to read requests from a file, and when finished return to reading requests from the input stream.

The prefix directive **once** can be placed in front of any **mmpmon** request. The **once** prefix indicates that the request be run only once, irrespective of the setting of the **-r** flag on the **mmpmon** command. It is useful for requests that do not need to be issued more than once, such as to set up the node list or turn on the request histogram facility.

These rules apply when using the **once** prefix directive and **source** request:

1. **once** with nothing after it is an error that terminates **mmpmon** processing.
2. A file invoked with the **source** request may contain **source** requests, causing file nesting of arbitrary depth.
3. No check is done for loops in the above situation.

4. The request **once source** *filename* causes the **once** prefix to be applied to all the **mmpmon** requests in *filename*, including any **source** requests in the file.
5. If a *filename* specified with the **source** request cannot be opened for read, an error is returned and **mmpmon** terminates.
6. If the **-r** flag on the **mmpmon** command has any value other than one, and all requests are prefixed with **once**, **mmpmon** runs all the requests once, issues a message, and then terminates.

An example of *once* and *source* usage

This example illustrates the use of **once** and **source**. This command is issued:

```
mmpmon -p -i command.file -r 0 -d 5000 | tee output.file
```

File **command.file** consists of this:

```
once source mmpmon.header
once rhist nr 512;1024;2048;4096 =
once rhist on
source mmpmon.commands
```

File **mmpmon.header** consists of this:

```
ver
reset
```

File **mmpmon.commands** consists of this:

```
fs_io_s
rhist s
```

The **output.file** is similar to this:

```
_ver_ _n_ 199.18.1.8 _nn_ node1 _v_ 2 _lv_ 4 _vt_ 0
_reset_ _n_ 199.18.1.8 _nn_ node1 _rc_ 0 _t_ 1129770129 _tu_ 511981
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ nr 512;1024;2048;4096 = _rc_ 0 _t_ 1129770131 _tu_ 524674
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ on _rc_ 0 _t_ 1129770131 _tu_ 524921
_fs_io_s_ _n_ 199.18.1.8 _nn_ node1 _rc_ 0 _t_ 1129770131 _tu_ 525062 _cl_ node1.localdomain
_fs_ gpfs1 _d_ 1 _br_ 0 _bw_ 0 _oc_ 0 _cc_ 0 _rdc_ 0 _wc_ 0 _dir_ 0 _iu_ 0
_fs_io_s_ _n_ 199.18.1.8 _nn_ node1 _rc_ 0 _t_ 1129770131 _tu_ 525062 _cl_ node1.localdomain
_fs_ gpfs2 _d_ 2 _br_ 0 _bw_ 0 _oc_ 0 _cc_ 0 _rdc_ 0 _wc_ 0 _dir_ 0 _iu_ 0
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ s _rc_ 0 _t_ 1129770131 _tu_ 525220 _k_ r
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ s _rc_ 0 _t_ 1129770131 _tu_ 525228 _k_ w
_end_
_fs_io_s_ _n_ 199.18.1.8 _nn_ node1 _rc_ 0 _t_ 1129770136 _tu_ 526685 _cl_ node1.localdomain
_fs_ gpfs1 _d_ 1 _br_ 0 _bw_ 0 _oc_ 0 _cc_ 0 _rdc_ 0 _wc_ 0 _dir_ 0 _iu_ 0
_fs_io_s_ _n_ 199.18.1.8 _nn_ node1 _rc_ 0 _t_ 1129770136 _tu_ 526685 _cl_ node1.localdomain
_fs_ gpfs2 _d_ 2 _br_ 0 _bw_ 395018 _oc_ 504 _cc_ 252 _rdc_ 0 _wc_ 251 _dir_ 0 _iu_ 147
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ s _rc_ 0 _t_ 1129770136 _tu_ 526888 _k_ r
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ s _rc_ 0 _t_ 1129770136 _tu_ 526896 _k_ w
_R_ 0 512 _NR_ 169
_L_ 0.0 1.0 _NL_ 155
_L_ 1.1 10.0 _NL_ 7
_L_ 10.1 30.0 _NL_ 1
_L_ 30.1 100.0 _NL_ 4
_L_ 100.1 200.0 _NL_ 2
_R_ 513 1024 _NR_ 16
_L_ 0.0 1.0 _NL_ 15
_L_ 1.1 10.0 _NL_ 1
_R_ 1025 2048 _NR_ 3
_L_ 0.0 1.0 _NL_ 32
_R_ 2049 4096 _NR_ 18
_L_ 0.0 1.0 _NL_ 18
_R_ 4097 0 _NR_ 16
_L_ 0.0 1.0 _NL_ 16
_end_
_fs_io_s_ _n_ 199.18.1.8 _nn_ node1 _rc_ 0 _t_ 1129770141 _tu_ 528613 _cl_ node1.localdomain
```

```
_fs_ gpfs1 _d_ 1 _br_ 0 _bw_ 0 _oc_ 0 _cc_ 0 _rdc_ 0 _wc_ 0 _dir_ 0 _iu_ 0
_fs_io_s_ _n_ 199.18.1.8 _nn_ node1 _rc_ 0 _t_ 1129770141 _tu_ 528613 _cl_ node1.localdomain
_fs_ gpfs2 _d_ 2 _br_ 0 _bw_ 823282 _oc_ 952 _cc_ 476 _rdc_ 0 _wc_ 474 _dir_ 0 _iu_ 459
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ s _rc_ 0 _t_ 1129770141 _tu_ 528812 _k_ r
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ s _rc_ 0 _t_ 1129770141 _tu_ 528820 _k_ w
_R_ 0 512 _NR_ 255
_L_ 0.0 1.0 _NL_ 241
_L_ 1.1 10.0 _NL_ 7
_L_ 10.1 30.0 _NL_ 1
_L_ 30.1 100.0 _NL_ 4
_L_ 100.1 200.0 _NL_ 2
_R_ 513 1024 _NR_ 36
_L_ 0.0 1.0 _NL_ 35
_L_ 1.1 10.0 _NL_ 1
_R_ 1025 2048 _NR_ 90
_L_ 0.0 1.0 _NL_ 90
_R_ 2049 4096 _NR_ 55
_L_ 0.0 1.0 _NL_ 55
_R_ 4097 0 _NR_ 38
_L_ 0.0 1.0 _NL_ 37
_L_ 1.1 10.0 _NL_ 1
_end_
_fs_io_s_ _n_ 199.18.1.8 _nn_ node1 _rc_ 0 _t_ 1129770146 _tu_ 530570 _cl_ node1.localdomain
_fs_ gpfs1 _d_ 1 _br_ 0 _bw_ 0 _oc_ 0 _cc_ 0 _rdc_ 0 _wc_ 0 _dir_ 0 _iu_ 1
_fs_io_s_ _n_ 199.18.1.8 _nn_ node1 _rc_ 0 _t_ 1129770146 _tu_ 530570 _cl_ node1.localdomain
_fs_ gpfs2 _d_ 2 _br_ 0 _bw_ 3069915 _oc_ 1830 _cc_ 914 _rdc_ 0 _wc_ 901 _dir_ 0 _iu_ 1070
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ s _rc_ 0 _t_ 1129770146 _tu_ 530769 _k_ r
_rhist_ _n_ 199.18.1.8 _nn_ node1 _req_ s _rc_ 0 _t_ 1129770146 _tu_ 530778 _k_ w
_R_ 0 512 _NR_ 526
_L_ 0.0 1.0 _NL_ 501
_L_ 1.1 10.0 _NL_ 14
_L_ 10.1 30.0 _NL_ 2
_L_ 30.1 100.0 _NL_ 6
_L_ 100.1 200.0 _NL_ 3
_R_ 513 1024 _NR_ 74
_L_ 0.0 1.0 _NL_ 70
_L_ 1.1 10.0 _NL_ 4
_R_ 1025 2048 _NR_ 123
_L_ 0.0 1.0 _NL_ 117
_L_ 1.1 10.0 _NL_ 6
_R_ 2049 4096 _NR_ 91
_L_ 0.0 1.0 _NL_ 84
_L_ 1.1 10.0 _NL_ 7
_R_ 4097 0 _NR_ 87
_L_ 0.0 1.0 _NL_ 81
_L_ 1.1 10.0 _NL_ 6
_end_
.............. and so forth ......................
```

If this command is issued with the same file contents:

```
mmpmon -i command.file -r 0 -d 5000 | tee output.file.english
```

The file **output.file.english** is similar to this:

```
mmpmon node 199.18.1.8 name node1 version 3.1.0
mmpmon node 199.18.1.8 name node1 reset OK
mmpmon node 199.18.1.8 name node1 rhist nr 512;1024;2048;4096 = OK
mmpmon node 199.18.1.8 name node1 rhist on OK
mmpmon node 199.18.1.8 name node1 fs_io_s OK
cluster:        node1.localdomain
filesystem:     gpfs1
disks:                  1
timestamp:      1129770175/950895
bytes read:             0
bytes written:          0
opens:                  0
closes:                 0
```

```
reads:                   0
writes:                  0
readdir:                 0
inode updates:           0

mmpmon node 199.18.1.8 name node1 fs_io_s OK
cluster:        node1.localdomain
filesystem:     gpfs2
disks:                   2
timestamp:      1129770175/950895
bytes read:              0
bytes written:

opens:                   0
closes:                  0
reads:                   0
writes:                  0
readdir:                 0
inode updates:           0
mmpmon node 199.18.1.8 name node1 rhist s OK read timestamp 1129770175/951117
mmpmon node 199.18.1.8 name node1 rhist s OK write timestamp 1129770175/951125
mmpmon node 199.18.1.8 name node1 fs_io_s OK
cluster:        node1.localdomain
filesystem:     gpfs1
disks:                   1
timestamp:      1129770180/952462
bytes read:              0
bytes written:           0
opens:                   0
closes:                  0
reads:                   0
writes:                  0
readdir:                 0
inode updates:           0

mmpmon node 199.18.1.8 name node1 fs_io_s OK
cluster:        node1.localdomain
filesystem:     gpfs2
disks:                   2
timestamp:      1129770180/952462
bytes read:              0
bytes written:      491310
opens:                 659
closes:                329
reads:                   0
writes:                327
readdir:                 0
inode updates:          74
mmpmon node 199.18.1.8 name node1 rhist s OK read timestamp 1129770180/952711
mmpmon node 199.18.1.8 name node1 rhist s OK write timestamp 1129770180/952720
size range               0 to        512 count        214
  latency range        0.0 to        1.0 count        187
  latency range        1.1 to       10.0 count         15
  latency range       10.1 to       30.0 count          6
  latency range       30.1 to      100.0 count          5
  latency range      100.1 to      200.0 count          1
size range             513 to       1024 count         27
  latency range        0.0 to        1.0 count         26
  latency range      100.1 to      200.0 count          1
size range            1025 to       2048 count         32
  latency range        0.0 to        1.0 count         29
  latency range        1.1 to       10.0 count          1
  latency range       30.1 to      100.0 count          2
size range            2049 to       4096 count         31
  latency range        0.0 to        1.0 count         30
  latency range       30.1 to      100.0 count          1
size range            4097 to          0 count         23
```

```
  latency range        0.0 to        1.0 count         23
mmpmon node 199.18.1.8 name node1 fs_io_s OK
cluster:        node1.localdomain
filesystem:     gpfs1
disks:                  1
timestamp:      1129770185/954401
bytes read:             0
bytes written:          0
opens:                  0
closes:                 0
reads:                  0
writes:                 0
readdir:                0
inode updates:          0

mmpmon node 199.18.1.8 name node1 fs_io_s OK
cluster:        node1.localdomain
filesystem:     gpfs2
disks:                  2
timestamp:      1129770185/954401
bytes read:             0
bytes written:     1641935
opens:                1062
closes:                531
reads:                   0
writes:                529
readdir:                 0
inode updates:         523
mmpmon node 199.18.1.8 name node1 rhist s OK read timestamp 1129770185/954658
mmpmon node 199.18.1.8 name node1 rhist s OK write timestamp 1129770185/954667
size range               0 to        512 count        305
  latency range        0.0 to        1.0 count        270
  latency range        1.1 to       10.0 count         21
  latency range       10.1 to       30.0 count          6
  latency range       30.1 to      100.0 count          6
  latency range      100.1 to      200.0 count          2
size range             513 to       1024 count         39
  latency range        0.0 to        1.0 count         36
  latency range        1.1 to       10.0 count          1
  latency range       30.1 to      100.0 count          1
  latency range      100.1 to      200.0 count          1
size range            1025 to       2048 count         89
  latency range        0.0 to        1.0 count         84
  latency range        1.1 to       10.0 count          2
  latency range       30.1 to      100.0 count          3
size range            2049 to       4096 count         56
  latency range        0.0 to        1.0 count         54
  latency range        1.1 to       10.0 count          1
  latency range       30.1 to      100.0 count          1
size range            4097 to          0 count         40
  latency range        0.0 to        1.0 count         39
  latency range        1.1 to       10.0 count          1
mmpmon node 199.18.1.8 name node1 fs_io_s OK
cluster:        node1.localdomain
filesystem:     gpfs1
disks:                  1
timestamp:      1129770190/956480
bytes read:             0
bytes written:          0
opens:                  0
closes:                 0
reads:                  0
writes:                 0
readdir:                0
inode updates:          0

mmpmon node 199.18.1.8 name node1 fs_io_s OK
```

```
cluster:        node1.localdomain
filesystem:     gpfs2
disks:                2
timestamp:      1129770190/956480
bytes read:           0
bytes written:     3357414
opens:                1940
closes:                969
reads:                   0
writes:                952
readdir:                 0
inode updates:        1101
mmpmon node 199.18.1.8 name node1 rhist s OK read timestamp 1129770190/956723
mmpmon node 199.18.1.8 name node1 rhist s OK write timestamp 1129770190/956732
size range              0 to        512 count        539
  latency range        0.0 to        1.0 count        494
  latency range        1.1 to       10.0 count         29
  latency range       10.1 to       30.0 count          6
  latency range       30.1 to      100.0 count          8
  latency range      100.1 to      200.0 count          2
size range            513 to       1024 count         85
  latency range        0.0 to        1.0 count         81
  latency range        1.1 to       10.0 count          2
  latency range       30.1 to      100.0 count          1
  latency range      100.1 to      200.0 count          1
size range           1025 to       2048 count        133
  latency range        0.0 to        1.0 count        124
  latency range        1.1 to       10.0 count          5
  latency range       10.1 to       30.0 count          1
  latency range       30.1 to      100.0 count          3
size range           2049 to       4096 count         99
  latency range        0.0 to        1.0 count         91
  latency range        1.1 to       10.0 count          6
  latency range       10.1 to       30.0 count          1
  latency range       30.1 to      100.0 count          1
size range           4097 to          0 count         95
  latency range        0.0 to        1.0 count         90
  latency range        1.1 to       10.0 count          4
  latency range       10.1 to       30.0 count          1
mmpmon node 199.18.1.8 name node1 fs_io_s OK
cluster:        node1.localdomain
filesystem:     gpfs1
disks:                1
timestamp:      1129770195/958310
bytes read:           0
bytes written:        0
opens:                0
closes:               0
reads:                0
writes:               0
readdir:              0
inode updates:        0

mmpmon node 199.18.1.8 name node1 fs_io_s OK
cluster:        node1.localdomain
filesystem:     gpfs2
disks:                2
timestamp:      1129770195/958310
bytes read:           0
bytes written:     3428107
opens:                2046
closes:               1023
reads:                   0
writes:                997
readdir:                 0
inode updates:        1321
mmpmon node 199.18.1.8 name node1 rhist s OK read timestamp 1129770195/958568
```

```
mmpmon node 199.18.1.8 name node1 rhist s OK write timestamp 1129770195/958577
size range               0 to        512 count        555
  latency range        0.0 to        1.0 count        509
  latency range        1.1 to       10.0 count         30
  latency range       10.1 to       30.0 count          6
  latency range       30.1 to      100.0 count          8
  latency range      100.1 to      200.0 count          2
size range             513 to       1024 count         96
  latency range        0.0 to        1.0 count         92
  latency range        1.1 to       10.0 count          2
  latency range       30.1 to      100.0 count          1
  latency range      100.1 to      200.0 count          1
size range            1025 to       2048 count        143
  latency range        0.0 to        1.0 count        134
  latency range        1.1 to       10.0 count          5
  latency range       10.1 to       30.0 count          1
  latency range       30.1 to      100.0 count          3
size range            2049 to       4096 count        103
  latency range        0.0 to        1.0 count         95
  latency range        1.1 to       10.0 count          6
  latency range       10.1 to       30.0 count          1
  latency range       30.1 to      100.0 count          1
size range            4097 to          0 count        100
  latency range        0.0 to        1.0 count         95
latency range         1.1 to         10.0 count          4
latency range        10.1 to         30.0 count          1
.............. and so forth ......................
```

Other information about mmpmon output

When interpreting the results from the **mmpmon** output there are several points to consider.

Consider these important points:

- On a node acting as a server of a GPFS file system to NFS clients, NFS I/O is accounted for in the statistics. However, the I/O is that which goes between GPFS and NFS. If NFS caches data, in order to achieve better performance, this activity is not recorded.
- I/O requests made at the application level may not be exactly what is reflected to GPFS. This is dependent on the operating system, and other factors. For example, an application read of 100 bytes may result in obtaining, and caching, a 1 MB block of data at a code level above GPFS (such as the libc I/O layer.) . Subsequent reads within this block result in no additional requests to GPFS.
- The counters kept by **mmpmon** are not atomic and may not be exact in cases of high parallelism or heavy system load. This design minimizes the performance impact associated with gathering statistical data.
- Reads from data cached by GPFS will be reflected in statistics and histogram data. Reads and writes to data cached in software layers above GPFS will be reflected in statistics and histogram data when those layers actually call GPFS for I/O.
- Activity from snapshots affects statistics. I/O activity necessary to maintain a snapshot is counted in the file system statistics.
- Some (generally minor) amount of activity in the root directory of a file system is reflected in the statistics of the file system manager node, and not the node which is running the activity.
- The open count also includes **creat()** call counts.

Counter sizes and counter wrapping

The **mmpmon** command may be run continuously for extended periods of time. The user must be aware that counters may wrap. This information applies to the counters involved:

- The statistical counters used for the **io_s** and **fs_io_s** requests are maintained by GPFS at all times, even when **mmpmon** has not been invoked. It is suggested that you use the **reset** request prior to starting a sequence of **io_s** or **fs_io_s** requests.
- The bytes read and bytes written counters are unsigned 64-bit integers. They are used in the **fs_io_s** and **io_s** requests, as the **_br_** and **_bw_** fields.
- The counters associated with the **rhist** requests are updated only when the request histogram facility has been enabled.
- The counters used in the **rhist** requests are unsigned 64-bit integers.
- All other counters are unsigned 32-bit integers.

Return codes from mmpmon

These are the return codes that can appear in the **_rc_** field:

0 Successful completion.

1 One of these has occurred:

1. For the **fs_io_s** request, no file systems are mounted.
2. For an **rhist** request, a request was issued that requires the request histogram facility to be enabled, but it is not. The facility is not enabled if:
   - Since the last **mmstartup** was issued, **rhist on** was never issued.
   - **rhist nr** was issued and **rhist on** was not issued afterwards.

2 For one of the **nlist** requests, the node name is not recognized.

13 For one of the **nlist** requests, the node name is a remote node, which is not allowed.

16 For one of the **rhist** requests, the histogram operations lock is busy. Retry the request.

17 For one of the **nlist** requests, the node name is already in the node list.

22 For one of the **rhist** requests, the size or latency range parameters were not in ascending order or were otherwise incorrect.

233 For one of the **nlist** requests, the specified node is not joined to the cluster.

668 For one of the **nlist** requests, quorum has been lost in the cluster.

Chapter 7. GPFS SNMP support

GPFS supports the use of the SNMP protocol for monitoring the status and configuration of the GPFS cluster. Using an SNMP application, the system administrator can get a detailed view of the system and be instantly notified of important events, such as a node or disk failure.

The Simple Network Management Protocol (SNMP) is an application-layer protocol that facilitates the exchange of management information between network devices. It is part of the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol suite. SNMP enables network administrators to manage network performance, find and solve network problems, and plan for network growth.

SNMP consists of commands to enumerate, read, and write managed variables that are defined for a particular device. It also has a **trap** command, for communicating events asynchronously.

The variables are organized as instances of objects, known as management information bases (MIBs). MIBs are organized in a hierarchical tree by organization (for example, IBM). A GPFS MIB is defined for monitoring many aspects of GPFS.

An SNMP agent software architecture typically consists of a master agent and a set of subagents, which communicate with the master agent through a specific agent/subagent protocol (the AgentX protocol in this case). Each subagent handles a particular system or type of device. A GPFS SNMP subagent is provided, which maps the SNMP objects and their values.

Installing Net-SNMP

The SNMP subagent runs on the collector node of the GPFS cluster. The collector node is designated by the system administrator.

Note: See “Collector node administration” on page 111 for information.

The Net-SNMP master agent (also called the SNMP daemon, or **snmpd**) must be installed on the collector node to communicate with the GPFS subagent and with your SNMP management application. Net-SNMP is included in most Linux distributions and should be supported by your Linux vendor. Source and binaries for several platforms are available at this Web site:

```
http://www.net-snmp.org/download.html
```

Note: Currently, the collector node must run on the Linux operating system. For an up-to-date list of supported operating systems, specific distributions, and other dependencies, refer to the GPFS FAQ at: publib.boulder.ibm.com/infocenter/clresctr/topic/com.ibm.cluster.gpfs.doc/gpfs_faqs/gpfsclustersfaq.html.

The GPFS subagent expects to find the following libraries:

```
libnetsnmpagent.so       -- from Net-SNMP
libnetsnmphelpers.so     -- from Net-SNMP
libnetsnmpmibs.so        -- from Net-SNMP
libnetsnmp.so            -- from Net-SNMP
libwrap.so               -- from TCP Wrappers
libcrypto.so             -- from OpenSSL
```

Note: TCP Wrappers and OpenSSL should have been prerequisited and installed when you installed Net-SMNP.

The installed libraries will be found in **/lib64** or **/usr/lib64** or **/usr/local/lib64** on 64-bit Linux systems. On 32-bit Linux systems, the libraries will usually be found in **/usr/lib** or **/usr/local/lib**. They may be installed under names like **libnetsnmp.so.5.1.2**. The GPFS subagent expects to find them without the appended

version information in the name. Library installation should create these symbolic links for you, so you will rarely need to create them yourself. You can ensure that symbolic links exist to the versioned name from the plain name. For example,

```
# cd /usr/lib64
# ln -s libnetsnmpmibs.so.5.1.2 libnetsnmpmibs.so
```

Repeat this for all the above listed libraries.

Note: For possible Linux platform and Net-SNMP version compatibility restrictions, see the GPFS README and the GPFS Frequently Asked Questions at: publib.boulder.ibm.com/infocenter/clresctr/topic/com.ibm.cluster.gpfs.doc/gpfs_faqs/gpfsclustersfaq.html.

Configuring Net-SNMP

The GPFS subagent process connects to the Net-SNMP master agent, **snmpd**.

The following entries are required in the **snmpd** configuration file on the collector node (usually, **/etc/snmp/snmpd.conf**):

```
master agentx
AgentXSocket tcp:localhost:705
trap2sink managementhost
```

where:

managementhost
: Is the host name or IP address of the host to which you want SNMP traps sent.

If your GPFS cluster has a large number of nodes or a large number of file systems for which information must be collected, you must increase the timeout and retry parameters for communication between the SNMP master agent and the GPFS subagent to allow time for the volume of information to be transmitted. The **snmpd** configuration file entries for this are:

```
agentXTimeout 60
agentXRetries 10
```

where:

agentXTimeout
: Is set to 60 seconds for subagent to master agent communication.

agentXRetries
: Is set to 10 for the number of communication retries.

Note: Other values may be appropriate depending on the number of nodes and file systems in your GPFS cluster.

After modifying the configuration file, restart the SNMP daemon.

Configuring management applications

To configure any SNMP-based management applications you might be using (such as Tivoli NetView® or Tivoli Netcool®, or others), you must make the GPFS MIB file available on the processor on which the management application runs.

You must also supply the management application with the host name or IP address of the collector node to be able to extract GPFS monitoring information through SNMP. To do this, you must be familiar with your SNMP-based management applications.

For more information about Tivoli NetView or Tivoli Netcool, see the documentation at the following information centers:

- http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp?toc=/com.ibm.itnetview.doc/toc.xml
- http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/index.jsp?toc=/com.ibm.netcool_precision.doc/toc.xml

Installing MIB files on the collector node and management node

The GPFS management information base (MIB) file is found on the collector node in the **/usr/lpp/mmfs/data** directory with the name **GPFS-MIB.txt**.

To install this file on the collector node, do the following:

1. Copy or link the **/usr/lpp/mmfs/data/GPFS-MIB.txt** MIB file into the **SNMP MIB** directory (usually, **/usr/share/snmp/mibs**).

   Alternatively, you could add the following line to the **snmp.conf** file (usually found in the directory **/etc/snmp**):

   ```
   mibdirs +/usr/lpp/mmfs/data
   ```

2. Add the following entry to the **snmp.conf** file (usually found in the directory **/etc/snmp**):

   ```
   mibs +GPFS-MIB
   ```

3. Restart the SNMP daemon.

Different management applications have different locations and ways for installing and loading a new MIB file. The following steps for installing the GPFS MIB file apply only to Net-SNMP. If you are using other management applications, such as NetView and NetCool, refer to corresponding product manuals (listed in "Configuring management applications" on page 110) for the procedure of MIB file installation and loading.

1. Remotely copy the **/usr/lpp/mmfs/data/GPFS-MIB.txt** MIB file from the collector node into the **SNMP MIB** directory (usually, **/usr/share/snmp/mibs**).
2. Add the following entry to the **snmp.conf** file (usually found in the directory **/etc/snmp**):

   ```
   mibs +GPFS-MIB
   ```

3. You might need to restart the SNMP management application. Other steps might be necessary to make the GPFS MIB available to your management application.

Collector node administration

Collector node administration includes: assigning, unassigning, and changing collector nodes. You can also see if a collector node is defined.

To assign a collector node and start the SNMP agent, enter:

```
mmchnode --snmp-agent -N NodeName
```

To unassign a collector node and stop the SNMP agent, enter:

```
mmchnode --nosnmp-agent -N NodeName
```

To see if there is a GPFS SNMP subagent collector node defined, enter:

```
mmlscluster [|grep snmp]
```

To change the collector node, issue the following two commands:

```
mmchnode --nosnmp-agent -N OldNodeName
```

```
mmchnode --snmp-agent -N NewNodeName
```

Starting and stopping the SNMP subagent

The SNMP subagent is started and stopped automatically.

The SNMP subagent is started automatically when GPFS is started on the collector node. If GPFS is already running when the collector node is assigned, the **mmchnode** command will automatically start the SNMP subagent.

The SNMP subagent is stopped automatically when GPFS is stopped on the node (**mmshutdown**) or when the SNMP collector node is unassigned (**mmchnode**).

The management and monitoring subagent

The GPFS SNMP management and monitoring subagent runs under an SNMP master agent such as Net-SNMP. It handles a portion of the SNMP OID space.

The management and monitoring subagent connects to the GPFS daemon on the collector node to retrieve updated information about the status of the GPFS cluster.

SNMP data can be retrieved using an SNMP application such as Tivoli NetView. NetView provides a MIB browser for retrieving user-requested data, as well as an event viewer for displaying asynchronous events.

Information that is collected includes status, configuration, and performance data about GPFS clusters, nodes, disks, file systems, storage pools, and asynchronous events. The following is a sample of the data that is collected for each of the following categories:

- Cluster status and configuration (see Table 20 on page 113 and Table 21 on page 113)
  - Name
  - Number of nodes
  - Primary and secondary servers
- Node status and configuration (see Table 22 on page 114 and Table 23 on page 114)
  - Name
  - Current status
  - Type
  - Platform
- File system status and performance (see Table 24 on page 115 and Table 25 on page 116)
  - Name
  - Status
  - Total space
  - Free space
  - Accumulated statistics
- Storage pools (see Table 26 on page 116)
  - Name
  - File system to which the storage pool belongs
  - Total storage pool space
  - Free storage pool space
  - Number of disks in the storage pool
- Disk status, configuration, and performance (see Table 27 on page 117, Table 28 on page 117, and Table 29 on page 118)
  - Name
  - Status
  - Total space
  - Free space
  - Usage (metadata/data)
  - Availability
  - Statistics

- Asynchronous events (traps) (see Table 30 on page 118)
  - File system mounted or unmounted
  - Disks added, deleted, or changed
  - Node failure or recovery
  - File system creation, deletion, or state change
  - Storage pool is full or nearly full

Note: If file systems are not mounted on the collector node at the time that an SNMP request is received, the subagent can still obtain a list of file systems, storage pools, and disks, but some information, such as performance statistics, will be missing.

SNMP object IDs

The management and monitoring SNMP subagent serves the OID space defined as **ibm.ibmProd.ibmGPFS**, which is the numerical **enterprises.2.6.212** OID space.

Underneath this top-level space are the following:

- **gpfsTraps** at **ibmGPFS.0**
- **gpfsMIBObjects** at **ibmGPFS.1**

MIB objects

gpfsMIBObjects provides a space of objects that can be retrieved using a MIB browser application. Net-SNMP provides the **snmpget**, **snmpgetnext**, **snmptable**, and **snmpwalk** commands, which can be used to retrieve the contents of these fields.

Cluster status information

Table 20 shows the current status information for the GPFS cluster:

Table 20. gpfsClusterStatusTable: Cluster status information

Value	Description
gpfsClusterName	The cluster name.
gpfsClusterId	The cluster ID.
gpfsClusterMinReleaseLevel	The currently enabled cluster functionality level.
gpfsClusterNumNodes	The number of nodes that belong to the cluster.
gpfsClusterNumFileSystems	The number of file systems that belong to the cluster.

Cluster configuration information

Table 21 shows the GPFS cluster configuration information:

Table 21. gpfsClusterConfigTable: Cluster configuration information

Value	Description
gpfsClusterConfigName	The cluster name.
gpfsClusterUidDomain	The UID domain name for the cluster.
gpfsClusterRemoteShellCommand	The remote shell command being used.
gpfsClusterRemoteFileCopyCommand	The remote file copy command being used.
gpfsClusterPrimaryServer	The primary GPFS cluster configuration server.
gpfsClusterSecondaryServer	The secondary GPFS cluster configuration server.

Table 21. gpfsClusterConfigTable: Cluster configuration information (continued)

Value	Description
gpfsClusterMaxBlockSize	The maximum file system block size.
gpfsClusterDistributedTokenServer	Indicates whether the distributed token server is enabled.
gpfsClusterFailureDetectionTime	The desired time for GPFS to react to a node failure.
gpfsClusterTCPPort	The TCP port number.

Node status information

Table 22 shows the collected status data for each node:

Table 22. gpfsNodeStatusTable: Node status information

Node	Description
gpfsNodeName	The node name used by the GPFS daemon.
gpfsNodeIp	The node IP address.
gpfsNodePlatform	The operating system being used.
gpfsNodeStatus	The node status (for example, up or down).
gpfsNodeFailureCount	The number of node failures.
gpfsNodeThreadWait	The longest hung thread's wait time (milliseconds).
gpfsNodeHealthy	Indicates whether the node is healthy in terms of hung threads. If there are hung threads, the value is no.
gpfsNodeDiagnosis	Shows the number of hung threads and detail on the longest hung thread.
gpfsNodeVersion	The GPFS product version of the currently running daemon.

Node configuration information

Table 23 shows the collected configuration data for each node:

Table 23. gpfsNodeConfigTable: Node configuration information

Node	Description
gpfsNodeConfigName	The node name used by the GPFS daemon.
gpfsNodeType	The node type (for example, manager/client or quorum/nonquorum).
gpfsNodeAdmin	Indicates whether the node is one of the preferred admin nodes.
gpfsNodePagePoolL	The size of the cache (low 32 bits).
gpfsNodePagePoolH	The size of the cache (high 32 bits).
gpfsNodePrefetchThreads	The number of prefetch threads.
gpfsNodeMaxMbps	An estimate of how many megabytes of data can be transferred per second.
gpfsNodeMaxFilesToCache	The number of inodes to cache for recently-used files that have been closed.
gpfsNodeMaxStatCache	The number of inodes to keep in the stat cache.

Table 23. gpfsNodeConfigTable: Node configuration information (continued)

Node	Description
gpfsNodeWorker1Threads	The maximum number of worker threads that can be started.
gpfsNodeDmapiEventTimeout	The maximum time the file operation threads will block while waiting for a DMAPI synchronous event (milliseconds).
gpfsNodeDmapiMountTimeout	The maximum time that the mount operation will wait for a disposition for the mount event to be set (seconds).
gpfsNodeDmapiSessFailureTimeout	The maximum time the file operation threads will wait for the recovery of the failed DMAPI session (seconds).
gpfsNodeNsdServerWaitTimeWindowOnMount	Specifies a window of time during which a mount can wait for NSD servers to come up (seconds).
gpfsNodeNsdServerWaitTimeForMount	The maximum time that the mount operation will wait for NSD servers to come up (seconds).
gpfsNodeMinMissedPingTimeout	The lower bound on missed ping timeout (seconds).
gpfsNodeMaxMissedPingTimeout	The upper bound on missed ping timeout (seconds).
gpfsNodeUnmountOnDiskFail	Indicates how the GPFS daemon will respond when a disk failure is detected. If it is ″true″, any disk failure will cause only the local node to forcibly unmount the file system that contains the failed disk.

File system status information

Table 24 shows the collected status information for each file system:

Table 24. gpfsFileSystemStatusTable: File system status information

Value	Description
gpfsFileSystemName	The file system name.
gpfsFileSystemStatus	The status of the file system.
gpfsFileSystemXstatus	The executable status of the file system.
gpfsFileSystemTotalSpaceL	The total disk space of the file system in kilobytes (low 32 bits).
gpfsFileSystemTotalSpaceH	The total disk space of the file system in kilobytes (high 32 bits).
gpfsFileSystemNumTotalInodesL	The total number of file system inodes (low 32 bits).
gpfsFileSystemNumTotalInodesH	The total number of file system inodes (high 32 bits).
gpfsFileSystemFreeSpaceL	The free disk space of the file system in kilobytes (low 32 bits).
gpfsFileSystemFreeSpaceH	The free disk space of the file system in kilobytes (high 32 bits).
gpfsFileSystemNumFreeInodesL	The number of free file system inodes (low 32 bits).
gpfsFileSystemNumFreeInodesH	The number of free file system inodes (high 32 bits).

File system performance information

Table 25 shows the file system performance information:

Table 25. gpfsFileSystemPerfTable: File system performance information

Value	Description
gpfsFileSystemPerfName	The file system name.
gpfsFileSystemBytesReadL	The number of bytes read from disk, not counting those read from cache (low 32 bits).
gpfsFileSystemBytesReadH	The number of bytes read from disk, not counting those read from cache (high 32 bits).
gpfsFileSystemBytesCacheL	The number of bytes read from the cache (low 32 bits).
gpfsFileSystemBytesCacheH	The number of bytes read from the cache (high 32 bits).
gpfsFileSystemBytesWrittenL	The number of bytes written, to both disk and cache (low 32 bits).
gpfsFileSystemBytesWrittenH	The number of bytes written, to both disk and cache (high 32 bits).
gpfsFileSystemReads	The number of read operations supplied from disk.
gpfsFileSystemCaches	The number of read operations supplied from cache.
gpfsFileSystemWrites	The number of write operations to both disk and cache.
gpfsFileSystemOpenCalls	The number of file system open calls.
gpfsFileSystemCloseCalls	The number of file system close calls.
gpfsFileSystemReadCalls	The number of file system read calls.
gpfsFileSystemWriteCalls	The number of file system write calls.
gpfsFileSystemReaddirCalls	The number of file system readdir calls.
gpfsFileSystemInodesWritten	The number of inode updates to disk.
gpfsFileSystemInodesRead	The number of inode reads.
gpfsFileSystemInodesDeleted	The number of inode deletions.
gpfsFileSystemInodesCreated	The number of inode creations.
gpfsFileSystemStatCacheHit	The number of stat cache hits.
gpfsFileSystemStatCacheMiss	The number of stat cache misses.

Storage pool information

Table 26 shows the collected information for each storage pool:

Table 26. gpfsStgPoolTable: Storage pool information

Value	Description
gpfsStgPoolName	The name of the storage pool.
gpfsStgPoolFSName	The name of the file system to which the storage pool belongs.
gpfsStgPoolTotalSpaceL	The total disk space in the storage pool in kilobytes (low 32 bits).
gpfsStgPoolTotalSpaceH	The total disk space in the storage pool in kilobytes (high 32 bits).

Table 26. gpfsStgPoolTable: Storage pool information (continued)

Value	Description
gpfsStgPoolFreeSpaceL	The free disk space in the storage pool in kilobytes (low 32 bits).
gpfsStgPoolFreeSpaceH	The free disk space in the storage pool in kilobytes (high 32 bits).
gpfsStgPoolNumDisks	The number of disks in the storage pool.

Disk status information

Table 27 shows the collected status information for each disk:

Table 27. gpfsDiskStatusTable: Disk status information

Value	Description
gpfsDiskName	The disk name.
gpfsDiskFSName	The name of the file system to which the disk belongs.
gpfsDiskStgPoolName	The name of the storage pool to which the disk belongs.
gpfsDiskStatus	The status of a disk (values: NotInUse, InUse, Suspended, BeingFormatted, BeingAdded, BeingEmptied, BeingDeleted, BeingDeleted-p, ReferencesBeingRemoved, BeingReplaced or Replacement).
gpfsDiskAvailability	The availability of the disk (Unchanged, OK, Unavailable, Recovering).
gpfsDiskTotalSpaceL	The total disk space in kilobytes (low 32 bits).
gpfsDiskTotalSpaceH	The total disk space in kilobytes (high 32 bits).
gpfsDiskFullBlockFreeSpaceL	The full block (unfragmented) free space in kilobytes (low 32 bits).
gpfsDiskFullBlockFreeSpaceH	The full block (unfragmented) free space in kilobytes (high 32 bits).
gpfsDiskSubBlockFreeSpaceL	The sub-block (fragmented) free space in kilobytes (low 32 bits).
gpfsDiskSubBlockFreeSpaceH	The sub-block (fragmented) free space in kilobytes (high 32 bits).

Disk configuration information

Table 28 shows the collected disk configuration information for each disk:

Table 28. gpfsDiskConfigTable: Disk configuration information

Value	Description
gpfsDiskConfigName	The disk name.
gpfsDiskConfigFSName	The name of the file system to which the disk belongs.
gpfsDiskConfigStgPoolName	The name of the storage pool to which the disk belongs.
gpfsDiskMetadata	Indicates whether the disk holds metadata.
gpfsDiskData	Indicates whether the disk holds data.

Disk performance information

Table 29 shows the collected disk performance information for each disk:

Table 29. gpfsDiskPerfTable: Disk performance information

Value	Description
gpfsDiskPerfName	The disk name.
gpfsDiskPerfFSName	The name of the file system to which the disk belongs.
gpfsDiskPerfStgPoolName	The name of the storage pool to which the disk belongs.
gpfsDiskReadTimeL	The total time spent waiting for disk read operations (low 32 bits).
gpfsDiskReadTimeH	The total time spent waiting for disk read operations (high 32 bits).
gpfsDiskWriteTimeL	The total time spent waiting for disk write operations in microseconds (low 32 bits).
gpfsDiskWriteTimeH	The total time spent waiting for disk write operations in microseconds (high 32 bits).
gpfsDiskLongestReadTimeL	The longest disk read time in microseconds (low 32 bits).
gpfsDiskLongestReadTimeH	The longest disk read time in microseconds (high 32 bits).
gpfsDiskLongestWriteTimeL	The longest disk write time in microseconds (low 32 bits).
gpfsDiskLongestWriteTimeH	The longest disk write time in microseconds (high 32 bits).
gpfsDiskShortestReadTimeL	The shortest disk read time in microseconds (low 32 bits).
gpfsDiskShortestReadTimeH	The shortest disk read time in microseconds (high 32 bits).
gpfsDiskShortestWriteTimeL	The shortest disk write time in microseconds (low 32 bits).
gpfsDiskShortestWriteTimeH	The shortest disk write time in microseconds (high 32 bits).
gpfsDiskReadBytesL	The number of bytes read from the disk (low 32 bits).
gpfsDiskReadBytesH	The number of bytes read from the disk (high 32 bits).
gpfsDiskWriteBytesL	The number of bytes written to the disk (low 32 bits).
gpfsDiskWriteBytesH	The number of bytes written to the disk (high 32 bits).
gpfsDiskReadOps	The number of disk read operations.
gpfsDiskWriteOps	The number of disk write operations.

Net-SNMP traps

Traps provide asynchronous notification to the SNMP application when a particular event has been triggered in GPFS. Table 30 shows the trap types that are defined:

Table 30. Net-SNMP traps

Net-SNMP trap type	This event is triggered by:
Mount	By the mounting node when the file system is mounted on a node.
Unmount	By the unmounting node when the file system is unmounted on a node.

Table 30. Net-SNMP traps (continued)

Net-SNMP trap type	This event is triggered by:
Add Disk	By the file system manager when a disk is added to a file system on a node.
Delete Disk	By the file system manager when a disk is deleted from a file system.
Change Disk	By the file system manager when the status of a disk or the availability of a disk is changed within the file system.
SGMGR Takeover	By the cluster manager when a file system manager takeover is successfully completed for the file system.
Node Failure	By the cluster manager when a node fails.
Node Recovery	By the cluster manager when a node recovers normally.
File System Creation	By the file system manager when a file system is successfully created.
File System Deletion	By the file system manager when a file system is deleted.
File System State Change	By the file system manager when the state of a file system changes.
New Connection	When a new connection thread is established between the events exporter and the management application.
Event Collection Buffer Overflow	By the collector node when the internal event collection buffer in the GPFS daemon overflows.
Token Manager Status	Every time the performance monitor thread wakes up (approximately every 30 seconds).
Hung Thread	By the affected node when a hung thread is detected. The GPFS Events Exporter Watchdog thread periodically checks for threads that have been waiting for longer than a threshold amount of time.
Storage Pool Utilization	By the file system manager when the utilization of a storage pool becomes full or almost full.

Chapter 8. Identity management on Windows

GPFS allows file sharing among AIX, Linux, and Windows nodes. AIX and Linux rely on 32-bit user and group IDs for file ownership and access control purposes, while Windows uses variable-length security identifiers (SIDs). The difference in the user identity description models presents a challenge to any subsystem that allows for heterogeneous file sharing.

GPFS uses 32-bit ID name space as the canonical name space, and Windows SIDs are mapped into this name space as needed. Two different mapping algorithms are used (depending on system configuration):

- GPFS built-in auto-generated mapping
- User-defined mappings stored in the Microsoft® Windows Active Directory using the Microsoft Identity Management for UNIX (IMU) component

Auto-generated ID mappings

Auto-generated ID mappings are the default. If no explicit mappings are created by the system administrator in the Active Directory using Microsoft Identity Management for UNIX (IMU), all mappings between security identifiers (SIDs) and UNIX IDs will be created automatically using a reserved range in UNIX ID space.

Unless the default reserved ID range overlaps with an ID already in use, no further configuration is needed to use the auto-generated mapping function. If you have a specific file system or subtree that are only accessed by user applications from Windows nodes (even if AIX or Linux nodes are used as NSD servers), auto-generated mappings will be sufficient for all application needs.

The default reserved ID range used by GPFS starts with ID 15,000,000 and covers 15,000,000 IDs. The reserved range should not overlap with any user or group ID in use on any AIX or Linux nodes. To change the starting location or the size of the reserved ID range, use the following GPFS configuration parameters:

sidAutoMapRangeLength
Controls the length of the reserved range for Windows SID to UNIX ID mapping.

sidAutoMapRangeStart
Specifies the start of the reserved range for Windows SID to UNIX ID mapping.

Note: For planning purposes, remember that auto-generated ID mappings are stored permanently with file system metadata. A change in the **sidAutoMapRangeStart** value is only effective for file systems created after the configuration change.

User-defined ID mappings

If you will be sharing the same subset of files among applications running on Windows, AIX, or Linux nodes, GPFS can exploit the Microsoft Identity Management for UNIX (IMU) service for mapping between Windows SIDs and UNIX user and group IDs.

IMU is an optional component of Microsoft Windows Server 2003R2 that can be installed on domain controllers. IMU gives the system administrator the ability to define a numeric ID for most domain and built-in users and groups. These mappings are stored in the Active Directory and can be queried using the standard AD API. GPFS will query the AD to find any mappings that are defined, and will use them to perform mappings between Windows SIDs and UNIX IDs to facilitate file sharing between Windows and UNIX. Using IMU with GPFS is entirely optional; if mappings are not defined for some users, or there are no mappings defined, GPFS will fall back to auto-generated mapping.

For the IMU ID mapping to work correctly, all GPFS nodes in a specified cluster must belong to the same Windows domain. IMU-based ID mapping is only applicable to domain accounts and not local computer accounts.

Installing Windows IMU

The Identity Mapping for UNIX (IMU) component needs to be installed on the primary domain controller, as well as on any backup domain controllers. It is not installed by default. There are two components that need to be installed in order for IMU to function correctly.

To begin the installation, insert Windows Server 2003 R2 CD 2 into the drive (or mount the corresponding ISO image), then follow these steps:

1. Click **Start** → **Control Panel** → **Add or Remove Programs**. The **Add or Remove Programs** window is displayed.
   a. Click **Add/Remove Windows Components**
   b. Click the **Active Directory Services** line, and click **Details...**
   c. Select the checkbox next to the **Identity Management for UNIX line**, and click **Details...**.
   d. Make sure the checkbox for the **Server for NIS line** is checked and the **Password Synchronization** component (which is not needed by GPFS) is unchecked.
2. Finish the installation using files from the Windows CD 2:
   a. Find the **ADMIN** folder on Windows CD 2, and run **IDMU.exe**. This step will complete the IMU installation.

Configuring ID mappings in IMU

To configure ID mappings in Microsoft Identity Management for UNIX (IMU), follow the steps in this procedure.

1. Click **Start** → **Control Panel**→ **Administrative Tools** → **Active Directory Users and Computers**.
2. Select the **Users** branch in the tree on the left under the branch for your domain to see the list of users and groups in this domain.
3. Double-click on any user or group line to bring up the Properties window. If IMU is set up correctly, there will be a **UNIX Attributes** tab as shown in Figure 11 on page 123:

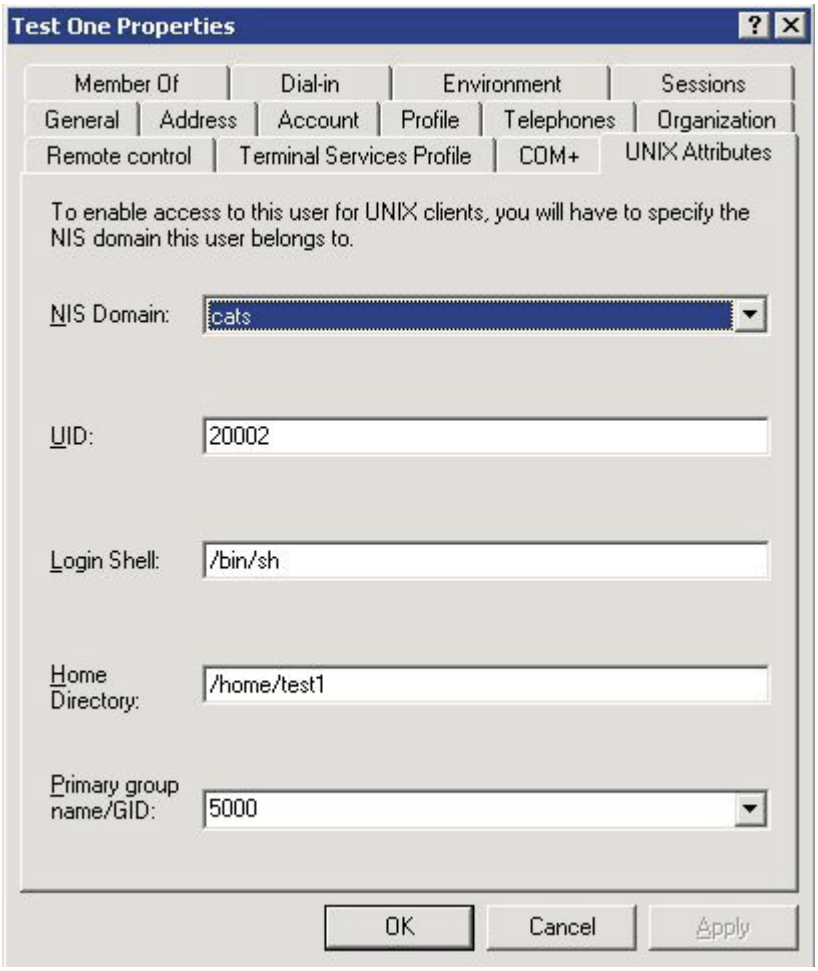


Figure 11. Properties window

Note: Because the IMU subsystem was originally designed to support integration with the UNIX Network Information Service (NIS), there is an **NIS Domain** field in the Properties window. You do not need to have NIS set up on the UNIX side. For GPFS, the NIS language does not matter.

Update information on the **UNIX Attributes** panel as follows:

1. Under the **NIS Domain** drop-down list, select the name of your Windows domain. Selecting **<none>** will remove an existing mapping.
2. Specify a UID in the **UID** field, and for Group objects, specify a GID. This will create a bidirectional mapping between the corresponding SID and a UNIX ID. IMU will disallow the use of the same UID or GID for more than one user or group to ensure that all mappings are unique. In addition to creating mappings for domain users and groups, you can create mappings for certain built-in accounts by going to the **Builtin branch** in the Active Directory Users and Computers panel.
3. Disregard the **Primary group name/GID** field because GPFS does not use it.

It is generally better to configure all ID mappings before mounting a GPFS file system for the first time. Doing that ensures that GPFS only stores properly remapped IDs on disk. However, it is possible to add or delete mappings at any time while GPFS file systems are mounted. GPFS picks up mapping changes dynamically (the code currently checks for mapping changes every 60 seconds), and will start using them at that time.

If an IMU mapping is configured for an ID that is already recorded in some file metadata, you must proceed with caution to avoid user confusion and access disruption. Auto-generated mappings already

stored in access control lists (ACLs) on disk will continue to map correctly to Windows SIDs. However, because the SID is now mapped to a different UNIX ID, when you access a file with an ACL containing its auto-generated ID, this access will effectively appear to GPFS as an access by a different user. Depending on the file access permissions, you might not be able to access files that were previously accessible. Rewriting affected ACLs after setting up a new mapping will help replace auto-generated IDs with IMU-mapped IDs, and will restore proper file access for the affected ID (this operation might need to be performed by the system administrator). Examining file ownership and permission information from a UNIX node (for example, using the **mmgetacl** command) is the easiest way to determine whether the ACL for a specified file contains auto-generated or IMU-mapped IDs.

Chapter 9. Miscellaneous advanced administration topics

There are miscellaneous advanced administration topics, including: how to change IP addresses, host names, and TCP/IP ports, how to enable the GPFS use of additional token servers and copying file system attributes from one cluster to another, and GPFS port usage information.

For more information, see:

Changing IP addresses and host names

GPFS assumes that IP addresses and host names remain constant. In the rare event that such a change becomes necessary or is inadvertently introduced by reinstalling a node with a disk image from a different node for example, follow the steps in this topic.

If all of the nodes in the cluster are affected and all the conditions in step 2 below are met:

1. Use the **mmshutdown -a** command to stop GPFS on all nodes.
2. Using the documented procedures for the operating system, add the new host names or IP addressees, but do not remove the old ones yet. This can be achieved, for example, by creating temporary alias entries in **/etc/hosts**. Avoid rebooting the nodes until the **mmchnode** command in step 3 is executed successfully. If any of these conditions cannot be met, utilize the alternate procedure described below.
3. Use **mmchnode --daemon-interface** and **--admin-interface** to update the GPFS configuration information.
4. If the IP addresses over which the subnet attribute is defined are changed, you need to update your configuration by using the **mmchconfig** command with the **subnets** attribute.
5. Start GPFS on all nodes with **mmstartup -a**.
6. Remove the unneeded old host names and IP addresses.

If only a subset of the nodes are affected, it may be easier to make the changes using these steps:

1. Before any of the host names or IP addresses are changed:
   - Use the **mmshutdown** command to stop GPFS on all affected nodes.
   - If the host names or IP addresses of the primary or secondary GPFS cluster configuration server nodes must change, use the **mmchcluster** command to specify another node to serve as the primary or secondary GPFS cluster configuration server.
   - If the host names or IP addresses of an NSD server node must change, temporarily remove the node from being a server with the **mmchnsd** command. Then, after the node has been added back to the cluster, use the **mmchnsd** command to change the NSDs to their original configuration. Use the **mmlsnsd** command to obtain the NSD server node names.
   - Use the **mmdelnode** command to delete all affected nodes from the GPFS cluster.
2. Change the node names and IP addresses using the documented procedures for the operating system.
3. If the IP addresses over which the subnet attribute is defined are changed, you need to update your configuration by using the **mmchconfig** command with the **subnets** attribute.
4. Issue the **mmaddnode** command to restore the nodes to the GPFS cluster.
5. If necessary, use the **mmchcluster** and **mmchnsd** commands to restore the original configuration and the NSD servers.

Using multiple token servers

Distributed locking in GPFS is implemented using token-based lock management. Associated with every lockable object is a token.

Before a lock on an object can be granted to a thread on a particular node, the lock manager on that node must obtain a token from the token server. Prior to GPFS 3.1, there was one token manager server per file system. With GPFS 3.1 and later, GPFS allows multiple token servers. The total number of token manager nodes depends on the number of manager nodes defined in the cluster.

When a file system is mounted initially, the file system manager is the only token server for the file system. Once the number of external mounts has reached an internal threshold, the file system manager appoints other manager nodes to share the token server load. Once the token state has been distributed, it remains distributed until all external mounts have gone away. The only nodes that are eligible to become token manager nodes are those designated as manager nodes.

The **maxFilesToCache** and **maxStatCache** parameters are indirectly affected by having multiple token manager nodes, because distributing the tokens across multiple nodes might allow keeping more tokens than with only one token server.

Exporting file system definitions between clusters

You can export a GPFS file system definition from one GPFS cluster to another.

To export file system definitions between clusters, follow these steps:

1. Ensure that all disks in all GPFS file systems to be migrated are in working order by issuing the **mmlsdisk** command. Verify that the disk status is ready and availability is up. If not, correct any problems and reissue the **mmlsdisk** command before continuing.
2. Stop all user activity in the file systems.
3. Follow any local administrative backup procedures to provide for protection of your file system data in the event of a failure.
4. Cleanly unmount all affected GPFS file systems. Do not use force unmount.
5. Export the GPFS file system definitions by issuing the **mmexportfs** command. This command creates the configuration output file *ExportDataFile* with all relevant file system and disk information. Retain this file as it is required when issuing the **mmimportfs** command to import your file systems into the new cluster. Depending on whether you are exporting a single file system or all of the file systems in the cluster, issue:

   ```
   mmexportfs fileSystemName -o ExportDataFile
   ```

   or

   ```
   mmexportfs all -o ExportDataFile
   ```

   Note:
6. Ensure that the file system disks from the old GPFS cluster are properly connected, and are online and available to be accessed from appropriate nodes of the new GPFS cluster.
7. To complete the movement of your file systems to the new cluster using the configuration file created in Step 5, issue one of these commands, depending on whether you are importing a single file system or all of the file systems in the cluster:

   ```
   mmimportfs fileSystemName -i ExportDataFile
   ```

   or

   ```
   mmimportfs all -i ExportDataFile
   ```

GPFS port usage

The nodes in a GPFS cluster communicate with each other using the TCP/IP protocol. The port number used by the main GPFS daemon (**mmfsd**) is controlled with the **tscTcpPort** configuration parameter. The default port number is 1191.

You can specify a different port number using the **mmchconfig** command:

```
mmchconfig tscTcpPort=PortNumber
```

When the main GPFS daemon (**mmfsd**) is not running on the primary and backup configuration server nodes, a separate service (**mmsdrserv**) is used to provide access to the configuration data to the rest of the nodes in the cluster. The port number used for this purpose is controlled with the **mmsdrservPort** parameter. By default, **mmsdrserv** uses the same port number as the one assigned to the main GPFS daemon. You can specify a different port number using the **mmchconfig** command:

```
mmchconfig mmsdrservPort=PortNumber
```

Certain commands (**mmadddisk**, **mmchmgr**, and so on) require an additional socket to be created for the duration of the command. The port numbers assigned to these temporary sockets are controlled with the **tscCmdPortRange** configuration parameter. If an explicit range is not specified, the port number is dynamically assigned by the operating system from the range of ephemeral port numbers. If you want to restrict the range of ports used by GPFS commands, use the **mmchconfig** command:

```
mmchconfig tscCmdPortRange=LowNumber-HighNumber
```

In a remote cluster setup, if GPFS on the remote cluster is configured to use a port number other than the default, you have to specify the port number to be used with the **mmremotecluster** command:

```
mmremotecluster update ClusterName -n tcpPort=PortNumber,Node,Node...
```

Table 31 provides GPFS port usage information:

Table 31. GPFS port usage

Descriptor	Explanation
Service provider	GPFS
Service name	mmfsd (mmfsd64) mmsdrserv
Port numbers	GPFS V3.2 - 1191 GPFS V3.1 - 1191 GPFS V2.3 and below - 6667 (mmfsd) 6668 (mmgetobj) 6669 (mmpmon) While executing certain commands, GPFS may need to create an additional socket, which gets a dynamic port number assigned by the operating system.
Protocols	TCP/IP
Source port range	The source port range is chosen by the operating system on the client side.
Is the service name/number pair in the default /etc/services file shipped with AIX and Linux distributions?	AIX: No Linux (SLES9, SLES8, RHEL3, RHEL4): No Linux (SLES10): Yes
Is the service name/number pair added to /etc/services by a product?	No

Table 31. GPFS port usage (continued)

Descriptor	Explanation
Binaries that listen on the ports	/usr/lpp/mmfs/bin/mmsdrserv /usr/lpp/mmfs/bin/mmfsd (Linux) /usr/lpp/mmfs/bin/aix32/mmfsd (AIX 32-bit kernel) /usr/lpp/mmfs/bin/aix64/mmfsd64 (AIX 64-bit kernel)

Can the service be configured to use a different port?	Yes. To change the main port used by GPFS, use:
	`mmchconfig tscTcpPort=PortNumber`
	To change the **mmsdrserv** port, use:
	`mmchconfig mmsdrservPort=PortNumber`
	To change the range of port numbers used for command execution, use:
	`mmchconfig tscCmdPortRange=LowNumber-HighNumber`
	To specify a port number when connecting to remote clusters, use the **mmremotecluster** command.

When is the service required? What depends on the service?	On the GPFS primary and secondary cluster configuration servers, either **mmsdrserv** or **mmfsd** needs to be running at all times to provide access to GPFS configuration data to the rest of the cluster. On other nodes, **mmfsd** must be running in order to mount a GPFS file system. Depending on the GPFS configuration, a node either has to be a member of the GPFS cluster or possess an authorized SSL key in order to establish a connection.
When the daemon starts and its port is already in use (for example, another resource has bound to it already), how does the daemon behave?	The daemon shuts down and tries to start over again. Most GPFS daemon down error messages are in the **mmfs.log.previous** log for the instance that failed. If the daemon restarted, it generates a new **mmfs.log.latest** log. Begin problem determination for these errors by examining the operating system error log. GPFS records file system or disk failures using the error logging facility provided by the operating system: **syslog** facility on Linux and **errpt** facility on AIX. See the *GPFS: Problem Determination Guide* for further information.
Is there an administrator interface to query the daemon and have it report its port number?	If the port number has been changed, the **mmlsconfig** command will display the information. If the command does not display a port number, it is set to the default value of 1191.

Is the service/port registered with the Internet Assigned Numbers Authority (IANA)?	Yes
	`gpfs 1191/tcp General Parallel File System`
	`gpfs 1191/udp General Parallel File System`
	`# Dave Craft <gpfs@ibm.com>`
	`November 2004`

Accessibility features for GPFS

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in GPFS:

- Keyboard-only operation
- Interfaces that are commonly used by screen readers
- Keys that are discernible by touch but do not activate just by touching them
- Industry-standard devices for ports and connectors
- The attachment of alternative input and output devices

The **IBM Cluster Information Center**, and its related publications, are accessibility-enabled. The accessibility features of the information center are described at http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic=/com.ibm.cluster.addinfo.doc/access.html.

Keyboard navigation

This product uses standard Microsoft Windows navigation keys.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility:

`http://www.ibm.com/able`

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. 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 product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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
Intellectual Property Law
Mail Station P300

2455 South Road,
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment or a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to the application programming interfaces for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at ″Copyright and trademark information″ at www.ibm.com/legal/copytrade.shtml

Intel®, Intel Inside® (logos), MMX and Pentium® are trademarks of Intel Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Red Hat, the Red Hat “Shadow Man” logo, and all Red Hat-based trademarks and logos are trademarks or registered trademarks of Red Hat, Inc., in the United States and other countries.

UNIX is a registered trademark of the Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Glossary

This glossary defines technical terms and abbreviations used in GPFS documentation. If you do not find the term you are looking for, refer to the index of the appropriate book or view the IBM Glossary of Computing Terms, located on the Internet at: http://www-306.ibm.com/software/globalization/terminology/index.jsp.

B

block utilization. The measurement of the percentage of used subblocks per allocated blocks.

C

cluster. A loosely-coupled collection of independent systems (nodes) organized into a network for the purpose of sharing resources and communicating with each other. See also *GPFS cluster*.

cluster configuration data. The configuration data that is stored on the cluster configuration servers.

cluster manager. The node that monitors node status using disk leases, detects failures, drives recovery, and selects file system managers. The cluster manager is the node with the lowest node number among the quorum nodes that are operating at a particular time.

control data structures. Data structures needed to manage file data and metadata cached in memory. Control data structures include hash tables and link pointers for finding cached data; lock states and tokens to implement distributed locking; and various flags and sequence numbers to keep track of updates to the cached data.

D

Data Management Application Program Interface (DMAPI). The interface defined by the Open Group's XDSM standard as described in the publication *System Management: Data Storage Management (XDSM) API Common Application Environment (CAE) Specification C429*, The Open Group ISBN 1-85912-190-X.

deadman switch timer. A kernel timer that works on a node that has lost its disk lease and has outstanding I/O requests. This timer ensures that the node cannot complete the outstanding I/O requests (which would risk causing file system corruption), by causing a panic in the kernel.

disk descriptor. A definition of the type of data that the disk contains and the failure group to which this disk belongs. See also *failure group*.

disposition. The session to which a data management event is delivered. An individual disposition is set for each type of event from each file system.

disk leasing. A method for controlling access to storage devices from multiple host systems. Any host that wants to access a storage device configured to use disk leasing registers for a lease; in the event of a perceived failure, a host system can deny access, preventing I/O operations with the storage device until the preempted system has reregistered.

domain. A logical grouping of resources in a network for the purpose of common management and administration.

F

failback. Cluster recovery from failover following repair. See also *failover*.

failover. (1) The process of transferring all control of the ESS to a single cluster in the ESS when the other cluster in the ESS fails. See also *cluster*. (2) The routing of all transactions to a second controller when the first controller fails. See also *cluster*. (3) The assumption of file system duties by another node when a node fails.

failure group. A collection of disks that share common access paths or adapter connection, and could all become unavailable through a single hardware failure.

fileset. A hierarchical grouping of files managed as a unit for balancing workload across a cluster.

file-management policy. A set of rules defined in a policy file that GPFS uses to manage file migration and file deletion. See also *policy*.

file-placement policy. A set of rules defined in a policy file that GPFS uses to manage the initial placement of a newly created file. See also *policy*.

file system descriptor. A data structure containing key information about a file system. This information includes the disks assigned to the file system (*stripe group*), the current state of the file system, and pointers to key files such as quota files and log files.

file system descriptor quorum. The number of disks needed in order to write the file system descriptor correctly.

file system manager. The provider of services for all the nodes using a single file system. A file system manager processes changes to the state or description of the file system, controls the regions of disks that are allocated to each node, and controls token management and quota management.

fragment. The space allocated for an amount of data too small to require a full block. A fragment consists of one or more subblocks.

G

GPFS cluster. A cluster of nodes defined as being available for use by GPFS file systems.

GPFS portability layer. The interface module that each installation must build for its specific hardware platform and Linux distribution.

GPFS recovery log. A file that contains a record of metadata activity, and exists for each node of a cluster. In the event of a node failure, the recovery log for the failed node is replayed, restoring the file system to a consistent state and allowing other nodes to continue working.

I

ill-placed file. A file assigned to one storage pool, but having some or all of its data in a different storage pool.

ill-replicated file. A file with contents that are not correctly replicated according to the desired setting for that file. This situation occurs in the interval between a change in the file's replication settings or suspending one of its disks, and the restripe of the file.

indirect block. A block containing pointers to other blocks.

IBM Virtual Shared Disk. The subsystem that allows application programs running on different nodes to access a logical volume as if it were local to each node. The logical volume is local to only one of the nodes (the server node).

inode. The internal structure that describes the individual files in the file system. There is one inode for each file.

J

journaled file system (JFS). A technology designed for high-throughput server environments, which are important for running intranet and other high-performance e-business file servers.

junction.

A special directory entry that connects a name in a directory of one fileset to the root directory of another fileset.

K

kernel. The part of an operating system that contains programs for such tasks as input/output, management and control of hardware, and the scheduling of user tasks.

L

logical volume. A collection of physical partitions organized into logical partitions, all contained in a single volume group. Logical volumes are expandable and can span several physical volumes in a volume group.

Logical Volume Manager (LVM). A set of system commands, library routines, and other tools that allow the user to establish and control logical volume (LVOL) storage. The LVM maps data between the logical view of storage space and the physical disk drive module (DDM).

M

metadata. A data structures that contain access information about file data. These include: inodes, indirect blocks, and directories. These data structures are not accessible to user applications.

metanode. The one node per open file that is responsible for maintaining file metadata integrity. In most cases, the node that has had the file open for the longest period of continuous time is the metanode.

mirroring. The process of writing the same data to multiple disks at the same time. The mirroring of data protects it against data loss within the database or within the recovery log.

multi-tailed. A disk connected to multiple nodes.

N

namespace. Space reserved by a file system to contain the names of its objects.

Network File System (NFS). A protocol, developed by Sun Microsystems, Incorporated, that allows any host in a network to gain access to another host or netgroup and their file directories.

Network Shared Disk (NSD). A component for cluster-wide disk naming and access.

NSD volume ID. A unique 16 digit hex number that is used to identify and access all NSDs.

node. An individual operating-system image within a cluster. Depending on the way in which the computer system is partitioned, it may contain one or more nodes.

node descriptor. A definition that indicates how GPFS uses a node. Possible functions include: manager node, client node, quorum node, and nonquorum node

node number. A number that is generated and maintained by GPFS as the cluster is created, and as nodes are added to or deleted from the cluster.

node quorum. The minimum number of nodes that must be running in order for the daemon to start.

node quorum with tiebreaker disks. A form of quorum that allows GPFS to run with as little as one quorum node available, as long as there is access to a majority of the quorum disks.

non-quorum node. A node in a cluster that is not counted for the purposes of quorum determination.

P

policy. A list of file-placement and service-class rules that define characteristics and placement of files. Several policies can be defined within the configuration, but only one policy set is active at one time.

policy rule. A programming statement within a policy that defines a specific action to be preformed.

pool. A group of resources with similar characteristics and attributes.

portability. The ability of a programming language to compile successfully on different operating systems without requiring changes to the source code.

primary GPFS cluster configuration server. In a GPFS cluster, the node chosen to maintain the GPFS cluster configuration data.

private IP address. A IP address used to communicate on a private network.

public IP address. A IP address used to communicate on a public network.

Q

quorum node. A node in the cluster that is counted to determine whether a quorum exists.

quota. The amount of disk space and number of inodes assigned as upper limits for a specified user, group of users, or fileset.

quota management. The allocation of disk blocks to the other nodes writing to the file system, and comparison of the allocated space to quota limits at regular intervals.

R

Redundant Array of Independent Disks (RAID). A collection of two or more disk physical drives that present to the host an image of one or more logical disk drives. In the event of a single physical device failure, the data can be read or regenerated from the other disk drives in the array due to data redundancy.

recovery. The process of restoring access to file system data when a failure has occurred. Recovery can involve reconstructing data or providing alternative routing through a different server.

replication. The process of maintaining a defined set of data in more than one location. Replication involves copying designated changes for one location (a source) to another (a target), and synchronizing the data in both locations.

rule. A list of conditions and actions that are triggered when certain conditions are met. Conditions include attributes about an object (file name, type or extension, dates, owner, and groups), the requesting client, and the container name associated with the object.

S

SAN-attached. Disks that are physically attached to all nodes in the cluster using Serial Storage Architecture (SSA) connections or using fibre channel switches

secondary GPFS cluster configuration server. In a GPFS cluster, the node chosen to maintain the GPFS cluster configuration data in the event that the primary GPFS cluster configuration server fails or becomes unavailable.

Secure Hash Algorithm digest (SHA digest). A character string used to identify a GPFS security key.

Serial Storage Architecture (SSA). An American National Standards Institute (ANSI) standard, implemented by IBM, for a high-speed serial interface that provides point-to-point connection for peripherals, such as storage arrays.

session failure. The loss of all resources of a data management session due to the failure of the daemon on the session node.

session node. The node on which a data management session was created.

Small Computer System Interface (SCSI). An ANSI-standard electronic interface that allows personal computers to communicate with peripheral hardware, such as disk drives, tape drives, CD-ROM drives, printers, and scanners faster and more flexibly than previous interfaces.

snapshot. A copy of changed data in the active files and directories of a file system with the exception of the inode number, which is changed to allow application programs to distinguish between the snapshot and the active files and directories.

source node. The node on which a data management event is generated.

SSA. See Serial Storage Architecture.

stand-alone client. The node in a one-node cluster.

storage area network (SAN). A dedicated storage network tailored to a specific environment, combining servers, storage products, networking products, software, and services.

storage pool. A grouping of storage space consisting of volumes, logical unit numbers (LUNs), or addresses that share a common set of administrative characteristics.

stripe group. The set of disks comprising the storage assigned to a file system.

striping. A storage process in which information is split into blocks (a fixed amount of data) and the blocks are written to (or read from) a series of disks in parallel.

subblock. The smallest unit of data accessible in an I/O operation, equal to one thirty-second of a data block.

system storage pool. A storage pool containing file system control structures, reserved files, directories, symbolic links, special devices, as well as the metadata associated with regular files, including indirect blocks and extended attributes The **system storage pool** can also contain user data.

T

token management. A system for controlling file access in which each application performing a read or write operation is granted some form of access to a specific block of file data. Token management provides data consistency and controls conflicts. Token management has two components: the token management server, and the token management function.

token management function. A component of token management that requests tokens from the token management server. The token management function is located on each cluster node.

token management server. A component of token management that controls tokens relating to the operation of the file system. The token management server is located at the file system manager node.

twin-tailed. A disk connected to two nodes.

U

user storage pool. A storage pool containing the blocks of data that make up user files.

V

virtual file system (VFS). A remote file system that has been mounted so that it is accessible to the local user.

virtual shared disk. See *IBM Virtual Shared Disk*.

virtual node (vnode). The structure that contains information about a file system object in an virtual file system (VFS).

Index

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

W

Y

Reader's comments - We'd like to hear from you

General Parallel File System
Advanced Administration Guide
Version 3 Release 2.1

Publication No. SC23-5182-02

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.
- Send your comments via e-mail to: mhvrcfs@us.ibm.com

If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	E-mail address

Readers' Comments — We'd Like to Hear from You
SC23-5182-02

IBM

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 58HA, Mail Station P181
2455 South Road
Poughkeepsie, NY
12601-5400

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

Program Number: 5724-N94
5765-G66

SC23-5182-02