



Michael Hennecke

# GPFS Multicluster with the IBM System Blue Gene Solution and eHPS Clusters

## Abstract

This IBM® Redpaper describes a case study in which an IBM System Blue Gene® Solution supercomputer is configured to natively access General Parallel File System (GPFS) 2.3 file systems that are owned by an IBM eServer™ pSeries® cluster with AIX 5L and the IBM eServer High Performance Switch (eHPS). The IBM System Blue Gene Solution service node (SN), front-end nodes (FENs), and I/O nodes (IONs) are configured as one GPFS cluster that does not contain any GPFS file systems. The pSeries cluster makes its GPFS file systems available to the Blue Gene system through the GPFS multicluster (GPFS-MC) functionality.

The study is performed in a customer environment where both systems are already operating in production: the Research Center Jülich (FZJ), Germany, operates a large POWER4+™ based AIX 5L cluster nicknamed “JUMP”, and an 8-rack Blue Gene system nicknamed “JUBL”. The JUMP cluster is already part of an existing wide-area GPFS multicluster setup—it is one of the DEISA.org sites. The Blue Gene system is added to this multicluster environment as an additional GPFS cluster.

# Introduction

GPFS is available for the IBM System Blue Gene Solution through a special-bid process: RPQ P91224, product number 5799-HCK. As outlined in the *GPFS HOWTO for BlueGene* that is shipped with the RPQ software, GPFS on the IBM System Blue Gene Solution is always implemented as a GPFS multicluster setup that includes two separate GPFS clusters:

- ▶ The SN, FEN(s) and Blue Gene IONs form one GPFS cluster, which itself does not contain any GPFS file systems but accesses the file systems of a second GPFS cluster. We call this cluster the *client cluster* or *local cluster*; it is named bgIO in the *GPFS HOWTO*.
- ▶ The file systems are provided by a second GPFS cluster, the *owning cluster* or *remote cluster*. This cluster is called gpfsNSD in the *GPFS HOWTO*. There are broadly two types of Blue Gene installations that exploit GPFS, which differ in the way the owning GPFS cluster is set up:

- Standalone Blue Gene systems

For these systems, the file servers are configured and installed together with the rest of the Blue Gene system. They are sized specifically for the Blue Gene environment, and are directly connected to the Blue Gene I/O network (called the *functional network* in the Blue Gene documentation), by one or more Gigabit Ethernet links per server.

- a. Blue Gene systems in heterogeneous environments

In these environments, the GPFS file systems reside on a separate cluster. This cluster typically has its own high-performance interconnect, and the GPFS file servers are sized for and connected to that interconnect. This case is more complex than the standalone case for two reasons:

- Network connectivity has to be established between the two clusters, whereas in the standalone case both clusters are on the same switched Ethernet network.
- Sizing of the file servers is not specific to Blue Gene and its (1 Gbps Ethernet based) I/O network. For example, the IBM eServer HPS provides 4 GByte/sec bidirectional bandwidth per link, normally with two links per node, and the GPFS servers on this interconnect are typically more powerful than those sized for standalone Blue Gene systems. This affects network design and tuning.

In this case study we describe the GPFS implementation in a heterogeneous environment, where the owning cluster is an IBM eServer pSeries cluster with AIX 5L and the eHPS interconnect. We discuss the extra planning and implementation steps that are needed for such a setup. Most of the specifics of this Redpaper do apply to both scenarios. It is intended to complement the *GPFS HOWTO for Blue Gene* by discussing a specific example for a typical supercomputing center.

In “Hardware setup” on page 3 we describe the hardware configuration of the clusters, including their physical connections. The software aspects of the network are covered in “Network configuration” on page 5. This includes hostnames, EtherChannel setup, and routing. These two sections are the prerequisites for “SSH setup for the Blue Gene GPFS cluster” on page 15, and “GPFS software installation” on page 20, which explain both how to create the Blue Gene GPFS cluster and how to establish the cross-cluster access. In “Tools for performance testing” on page 34 we present the steps that were taken to verify the performance of the solution, and our case study is wrapped up in “Summary” on page 42.

## Hardware setup

The hardware configuration of the clusters evolved over time, with three main steps shown from left to right in Figure 1 on page 4. Here we briefly describe the overall hardware setup; configuration details are discussed in the sections that follow.

1. The pSeries cluster (nicknamed “JUMP”) was installed with AIX 5L and the eServer HPS switch as its high-speed interconnect. Each node (LPAR) has two links into the eHPS network. The IP name of a node’s interface into the eHPS network is its hostname with a suffix of “f”. Most nodes are compute nodes, like node cn01 with its eHPS interface cn01f. In addition, there are four disk server nodes with Fiber Channel (FC) connections to disk storage subsystems, with IP names nsd01f, nsd02f, nsd03f, and nsd04f on the eHPS network. Not shown are the login nodes (with network connections to a user LAN) and a few other server nodes. GPFS on this cluster started with version 2.1, and uses the Virtual Shared Disk (VSD) layer over the eHPS switch. When the cluster was migrated to GPFS Version 2.3, Network Shared Disk (NSD) devices were configured on top of the existing VSDs.

**Note:** Make sure that you define primary and backup NSD servers for the NSDs that are created on top of the VSDs. Although this is not necessary *within* the eHPS cluster (because the underlying VSDs have primary and backup VSD servers defined), you will observe very strange errors when trying to access these NSDs from *remote* clusters. You can use the `mm1nsd -f <filesystem>` command to verify that NSD servers are configured.

2. When the cluster joined the DEISA Grid (see [www.deisa.org](http://www.deisa.org)) with its GPFS multiclusterc infrastructure, all nodes in the eHPS cluster needed TCP/IP connectivity to the other clusters on that Wide Area Network (WAN). This is discussed in detail later on.

There were two options for achieving this connectivity: Route this traffic through a gateway node *within* the pSeries/eHPS cluster, or connect all nodes to an additional *external* switch, which then routes the traffic to the WAN. The second approach was chosen. Each node received a 1 Gbps Ethernet adapter; the IP name of that interface is the hostname plus a suffix “m”. These are connected to a *DEISA GigaBit switch/router* shown in the center of Figure 1 on page 4. Routing is discussed in “Routing setup for cross-cluster connectivity” on page 12.

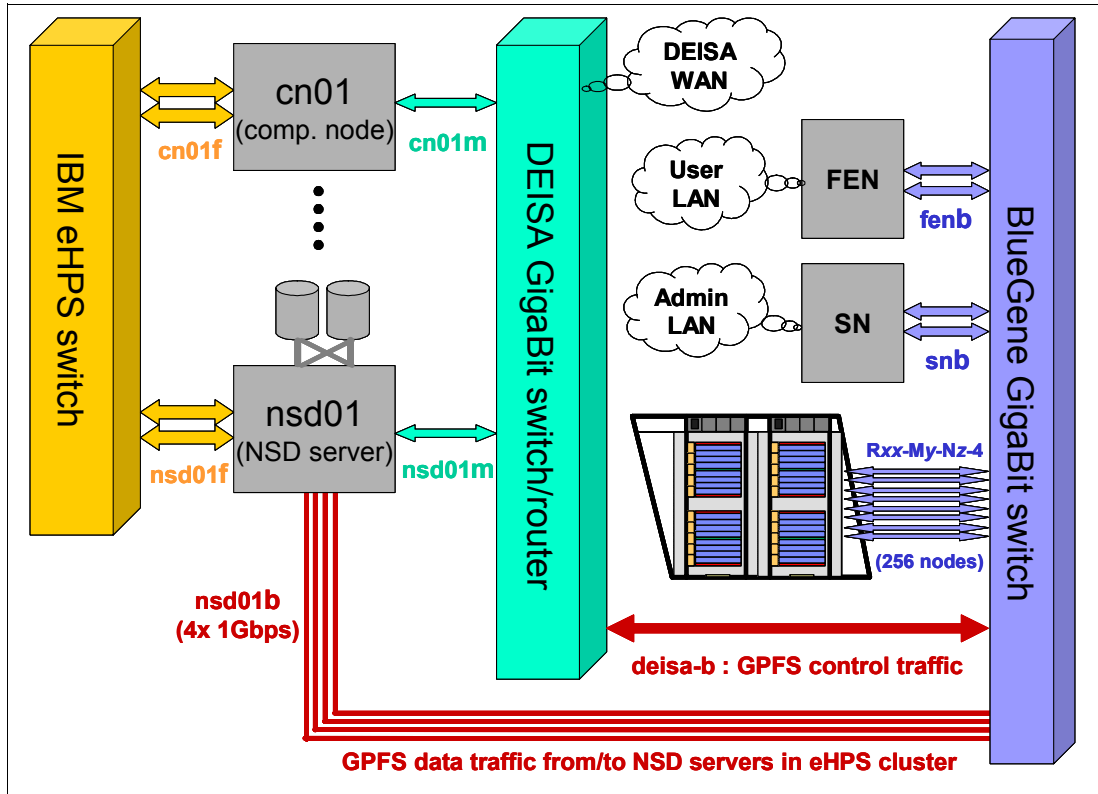


Figure 1 Networking setup of the pSeries/eHPS and BlueGene clusters

3. Recently, an 8-rack IBM System Blue Gene Solution was installed, nicknamed “JUBL”. The I/O network of Blue Gene is a 1 Gbps Ethernet network. Each Blue Gene I/O node has a single copper 1 Gbps port, and all IONs are connected to a *Blue Gene GigaBit switch* shown on the right of Figure 1. The machine’s service node (SN) and front-end node (FEN) are also on that network, and since they are external IBM OpenPower™ servers, they could be configured with multiple 1 Gbps adapters. SN and FEN also have connections to a user LAN and an administrative LAN, which are not important for the current discussion.
4. To make the GPFS file systems of the pSeries/eHPS cluster available to the Blue Gene cluster, all nodes on one side again needed TCP/IP connectivity to all nodes on the other side. As for the DEISA setup, there were several options to implement this. The solution that was chosen was twofold:
  - a. Connect the *DEISA switch/router* to the *Blue Gene switch*. Connectivity of all nodes on both sides can then be obtained by a suitable routing configuration, but bandwidth is limited by the inter-switch link. This link is used for GPFS *control* traffic.
  - b. To improve the bandwidth of GPFS *data* traffic, the four NSD server nodes in the pSeries/eHPS cluster were *directly* connected into the Blue Gene switch. On each node, four 1 Gbps links combined into a 4x EtherChannel were added, and connected to the *Blue Gene switch*. The bandwidth of this setup roughly matches the disk bandwidth available from the storage subsystems.

These two additional connections are shown on the bottom of Figure 1. The IP interfaces on the Blue Gene network are indicated by a “b” suffix, except for the IONs, which have only one network interface and where we follow a Blue Gene “physical location code” naming convention.

In the following sections, we discuss the networking configuration steps based on the above hardware setup.

## Network configuration

In this section we discuss the following topics:

- ▶ The setup of hostnames for the IONs
- ▶ Configuring EtherChannel (called channel bonding under Linux®) to increase the bandwidth of the pSeries servers' Ethernet connections
- ▶ Establishing routing between all the components
- ▶ Configuring various network services on the IONs

## IP names and addresses of the Blue Gene I/O nodes

Blue Gene uses a Gigabit Ethernet network as its I/O network (called the *functional network* in the Blue Gene documentation). The SN, FEN(s), and IONs are all directly connected into this network. Here we use a *supernet* consisting of four consecutive class-C networks to be able to accommodate 512 IONs and the external servers, even though we currently use only  $256+32=288$  IONs as detailed below. This I/O network is 222.444.666/22.

We strongly recommend that the Blue Gene IONs get their IP addresses on the Blue Gene I/O network statically assigned, and also have meaningful IP names set up. There are three places where this information needs to be stored:

- ▶ The /discovery/runPopIpPool script populates the DB2® table Bg1IpPool during installation. It is possible to specify Blue Gene physical location codes for the IP addresses as follows:

```
db2 "INSERT INTO Bg1IpPool (location,machineserialnumber,ipAddress)
VALUES('R00-M0-N0-I:J18-U11','BGL','222.444.666.33')"
```

```
db2 "INSERT INTO Bg1IpPool (location,machineserialnumber,ipAddress)
VALUES('R00-M0-N0-I:J18-U01','BGL','222.444.666.34')"
```

...

If you have not assigned physical location codes to the IP addresses when initially setting up this DB2 table, you should still be able to add them later. Refer to the first part of the "Support for Subnets" section in the *ionode.README* for details.

**Note:** These steps are: (1) delete the Bg1IpPool table, (2) recreate it with the desired settings, (3) set the IpAddress field in the Bg1Node table to NULL, and (4) re-run **PostDiscovery**.

- ▶ The /etc/hosts file. We add IP addresses, and long and short IP names as follows:

```
222.444.666.33 R00-M0-N0-3.your-domain.org R00-M0-N0-3
222.444.666.34 R00-M0-N0-4.your-domain.org R00-M0-N0-4
...
```

You need to update the hosts file in the following places:

- a. The /etc/hosts file on the SN and on the FEN(s).  
We recommend that you generate it on the SN, and then copy it to the FEN(s).
- b. The IONs also need access to a copy of the /etc/hosts file.  
On the SN, copy it to the location in /bgl/ where the startup rc-scripts expect it:

```
root@SN> cp -p /etc/hosts /bgl/dist/etc/hosts
```

- c. The `/etc/hosts` files on *all* nodes of the pSeries cluster (which owns the GPFS file systems) would also need to be updated, or you may rely on DNS on the pSeries side. We prefer to use DNS on the remote cluster for easier maintainability.
- DNS. The Linux `host` command always queries DNS, not the local `/etc/hosts` file (it does not honor the `/etc/nsswitch.conf` or `/etc/host.conf` settings). Since `mmaddnode` invokes the `host` command, all IONs must be known to DNS to enable them to be added to the GPFS cluster.

**Note:** It is beneficial to use hostnames that can easily be written with wildcards. An example is the creation of SSH *known hosts* files, which is discussed below.

In a Blue Gene *node card*, location J19 is the leftmost I/O card and J18 is the rightmost I/O card, each with two I/O nodes, U01 and U11. Figure 2 shows the RJ45 Ethernet connectors labeled “1” to “4” on the front of a node card, corresponding to I:J19-U11, I:J19-U01, I:J18-U11 and I:J18-U01. We prefer to use the node card location code plus the Ethernet connector number as hostnames.

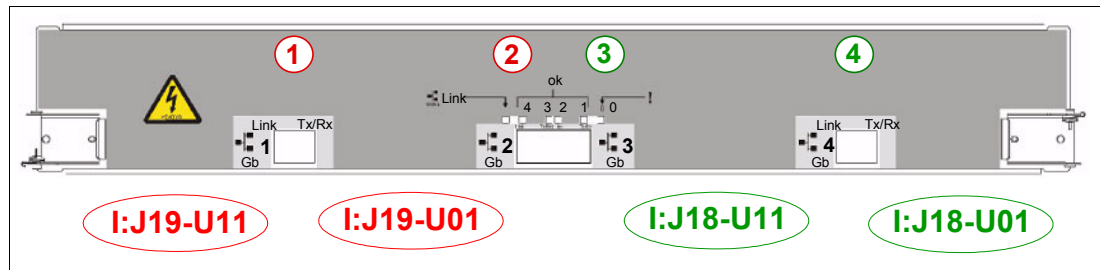


Figure 2 Ethernet ports on a Blue Gene node card

As shown in Figure 1 on page 4, the SN, the FEN, and the four NSD servers of the pSeries cluster are also on the Blue Gene I/O network, and these interfaces should also be added to the `/etc/hosts` file and DNS:

```
222.444.666.202 nsd01b.your-domain.org nsd01b
222.444.666.203 nsd02b.your-domain.org nsd02b
222.444.666.205 nsd03b.your-domain.org nsd03b
222.444.666.206 nsd04b.your-domain.org nsd04b
222.444.666.220 SN-b.your-domain.org SN-b
222.444.666.221 FEN-b.your-domain.org FEN-b
```

Finally, the DEISA switch/router has a leg into the Blue Gene I/O network. This interface is assigned the IP address 222.444.666.1 and an IP name of `deisa-b`.

## EtherChannel (or channel bonding) setup

The predominant I/O pattern in a Blue Gene system is hundreds of IONs with a single 1 Gbps Ethernet link accessing a high-bandwidth GPFS file system served by a pool of relatively few GPFS/NSD server nodes. For balanced performance, these NSD servers need more than one gigabit of network bandwidth. There are two options for achieving this:

- Use 10 Gbps adapters, which have a higher bandwidth.
- Combine multiple 1 Gbps links into an EtherChannel (bonding device under Linux).

Based on the available hardware (POWER4+ servers with their I/O drawers; Ethernet adapters; switch linecards), we decided to use the second approach because a 10 Gbps

solution would have been more expensive with only slightly better performance. On the NSD servers, we combine four 1 Gbps Ethernet links into an AIX 5L EtherChannel. This roughly matches the bandwidth that can be delivered by the storage attached to the NSD server (and could be increased to eight links per EtherChannel if needed). This configuration is shown in the left half of Figure 3 on page 7.

On the SN and FEN(s), we create a Linux bonding device with two links. This is useful if users want to quickly copy data into and out of the GPFS file systems on these nodes. The steps described below can also be used as a reference when the NSD servers are running Linux, which is typical for standalone Blue Gene systems. The right half of Figure 3 shows the SN and FEN(s) accessing the NSD servers.

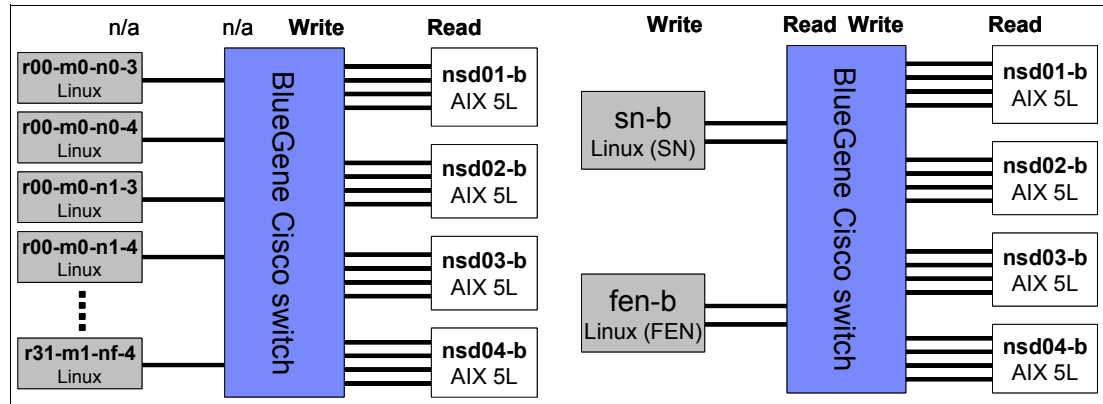


Figure 3 Ethernet channel bonding: IONs to NSD servers (left); SN and FEN to NSD servers (right)

We also indicate in Figure 3 which ports of the servers/switches need to be configured as EtherChannels, separately for file system *read* and *write* operations:

- ▶ ION *read* is controlled by the AIX 5L 4x EtherChannel setup of the NSD server.
- ▶ ION *write* is controlled by the Cisco/CatOS setup of the 4x channel to the NSD server.
- ▶ FEN *read* is controlled by:
  - The AIX 5L 4x EtherChannel setup of the NSD server, *and*
  - The Cisco/CatOS setup of the 2x EtherChannel to the FEN
- ▶ FEN *write* is controlled by:
  - The Linux/SLES9 2x channel bonding setup of the FEN, *and*
  - The Cisco/CatOS setup of the 4x EtherChannel to the NSD server
- ▶ The SN setup corresponds to the FEN setup.

Before describing the configuration steps in detail, some general remarks are useful as a motivation for our configuration. Channel bonding generally serves two purposes:

- ▶ Increasing *availability* by providing multiple, redundant paths between endpoints.
- ▶ Increasing *bandwidth* by utilizing multiple links in parallel. This can be further divided into two very different usage scenarios:
  - a. Single-stream performance, where a single connection needs higher bandwidth than one Ethernet link can provide
  - b. Aggregate performance, where multiple streams in parallel use more than one link

Here we only focus on bandwidth, and the gain in availability is a welcome side effect. Ignoring all modes that are specifically used for high availability, there are two general modes of operation that address the two usage scenarios:

- a. *Round-robin* distribution of Ethernet packets across *all* (active) ports in a channel.
- b. *Deterministic* forwarding of Ethernet packets using a *hash function*: All packets of a connection are forwarded to the *same* (active) port in a channel, which is determined by a hash function that may use source and destination MAC address, IP address, port number, or any combination thereof.

Obviously, only the round-robin mode can improve single-stream performance. But it also carries the risk that packets arrive out-of-order, which in the worst case would cause TCP/IP congestion control to slow down the connection after excessive retransmits. In addition, round-robin mode is not supported on the Cisco switches used here. So we decided to not use round-robin mode at all, even though it may be beneficial for single-stream file accesses from the FEN(s).

The more important case is a parallel job that uses many IONs to access GPFS, so there are many parallel connections to each of the NSD servers (assuming large block sequential I/O with files that are larger than 4x the GPFS file system block size, so the files will be striped across all four NSD servers). As each of the IONs has only a single 1 Gbps connection anyway, it is the aggregate performance that matters and a deterministic channeling mode is appropriate. Such a mode will always deliver packets in order, reducing the likelihood of congestion. Note that the three different platforms (AIX 5L, SLES9 Linux, and Cisco CatOS) have slightly different options to set up the hash function. This is described below.

## Setting up EtherChannel on the AIX 5L NSD servers

EtherChannel setup for AIX 5L is described in *AIX System Management Guide: Communications and Networks*, section “EtherChannel and IEEE 802.3ad Link Aggregation”. It can be found in the pSeries Information center at:

[http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.doc/aixbman/commadm/tcp\\_etherchannel.htm](http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.doc/aixbman/commadm/tcp_etherchannel.htm)

When data is *read* from GPFS, the AIX 5L configuration of the EtherChannel controls how packets are sent out. There are two parameters, *mode* and *hashmode*, that control how the EtherChannel operates. There are three *mode* settings:

- ▶ `mode=round_robin` (requires `hashmode=default`)  
This mode fully utilizes all links in the channel by sending packets in a round-robin mode across all links. It is good for back-to-back connections of two servers by an EtherChannel, but in our scenario it is problematic as described above.
- ▶ `mode=standard`  
This mode guarantees in-order delivery of packets, but may not optimally utilize the available bandwidth. It sends all packets for a single connection over the same link in the EtherChannel. The *hash\_mode* parameter controls the detailed algorithm of selecting this link:
  - `hashmode=default`  
Divide the lowest byte of the destination IP address (MAC address for non-IP traffic) by the number of (active) links in the channel. The remainder is the selected link number.
  - `hashmode=src_dst_port`  
Add the source and destination port numbers of the TCP connection, divide it by 2, truncate to integer, and divide by the number of (active) links in the channel. Again, the remainder is the selected link number.



- hashmode=src\_port and hashmode=dst\_port  
Same as src\_dst\_port but using only one of the two ports in the calculation.

Note that the adapters are enumerated in the order they are listed in the *adapter\_names* attribute when configuring the EtherChannel, not by their ethX number.

- mode=8023ad is similar to standard mode, except that it enables the use of the IEEE 802.3ad Link Aggregation Control Protocol (LACP) for automatic link aggregation.

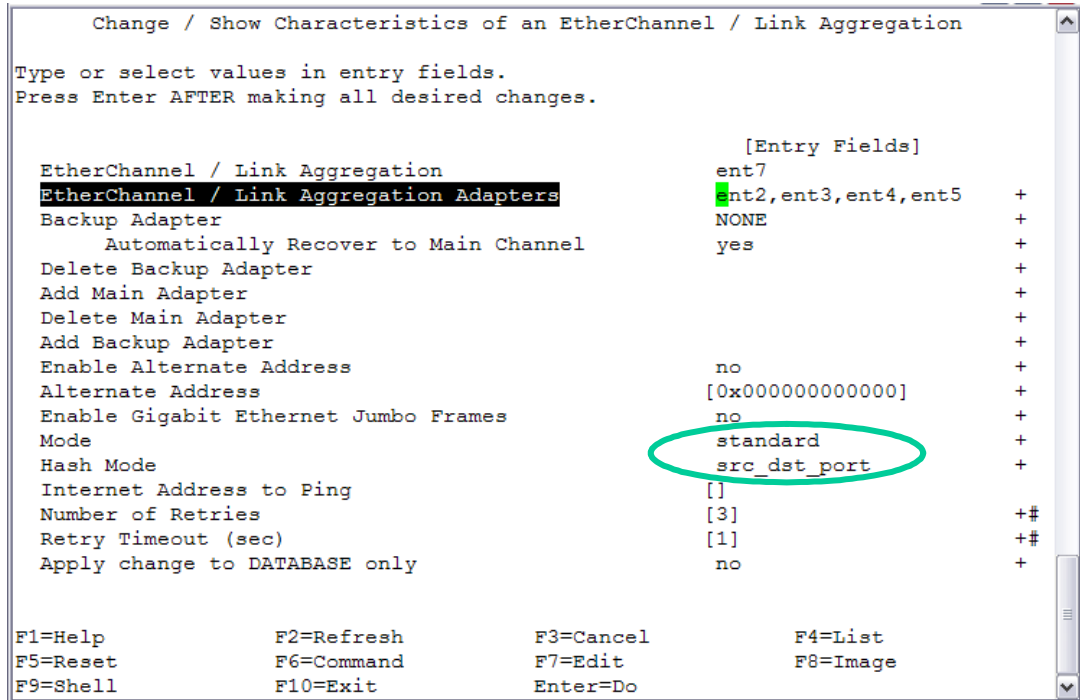


Figure 4 AIX 5L EtherChannel configuration with smitty

As shown in Figure 4, we used the **smitty etherchannel** command to configure the channel, using standard mode and a hashmode of *src\_dst\_port*. For our usage scenario, where the AIX 5L LPAR owns the file systems, the *source* port will always be 1191 (the port the mmfsd daemon listens on). But using both the source and destination port also supports the reverse case where the AIX 5L side accesses a file system on a remote cluster (in which case the *destination* port would always be 1191). Note that the channel mode and hashmode can be conveniently changed while the channel is active. The adapters that constitute the channel are listed in the *adapter\_names* attribute, as a list of ethX device names. To map the device names to physical location codes, you can use the AIX 5L **lscfg|grep ent** command:

```
sysadmin@nsd01> lscfg|grep ent|sort
+ ent0          U1.5-P1-I1/E1  10/100 Mbps Ethernet PCI Adapter II
(1410ff01)
+ ent1          U1.5-P1-I6/E1  Gigabit Ethernet-SX PCI-X Adapter (14106802)
+ ent2          U1.1-P1-I5/E2  2-Port 10/100/1000 Base-TX PCI-X Adapter
(14108902)
+ ent3          U1.1-P1-I5/E1  2-Port 10/100/1000 Base-TX PCI-X Adapter
(14108902)
+ ent4          U1.1-P1-I7/E2  2-Port 10/100/1000 Base-TX PCI-X Adapter
(14108902)
+ ent5          U1.1-P1-I7/E1  2-Port 10/100/1000 Base-TX PCI-X Adapter
(14108902)
```

```
+ ent6                U1.1-P1-I9/E1    10 Gigabit-LR Ethernet PCI-X Adapter
(1410bb02)
```

Here we used the four ports on the two adapters in locations U1.1-P1-I5 and U1.1-P1-I7.

## Setting up Linux bonding on the SN and FEN(s)

Linux provides channeling through the bonding driver, which is loaded as a module. The primary documentation is the *Linux Ethernet Bonding Driver HOWTO*. The driver has two modes of operation that are relevant to this study:

- ▶ `mode=balance-xor`:  
This is the Linux version of deterministic packet forwarding based on a hash function. Linux uses an XOR policy: the port is calculated based on (source MAC address XOR'd with destination MAC address) modulo slave port count. This selects the same slave port for each destination MAC address.
- ▶ `mode=balance-rr`:  
This is the Linux version of round-robin mode. We do not use it, but in case you plan to use `balance-rr`, the *Bonding Driver HOWTO* has some tips on how to adjust TCP/IP's congestion limits. It also contains a warning that for TCP traffic, you should not expect more than 2.3x worth of bandwidth in a `balance-rr` channel even if more links are added.

The bonding mode can be set as an option to the bonding module. You should also set the `miimon` option, which controls the time intervals in which the kernel checks link status. The *Bonding Driver HOWTO* recommends to use `miimon=100` and `mode=balance-rr`. As explained above we prefer `miimon=100` and `mode=balance-xor`. Here are the contents of our `bond0` configuration file on the SN, similar on the FEN(s):

```
sysadmin@SN> cat /etc/sysconfig/network/ifcfg-bond0

BOOTPROTO='static'
BROADCAST='222.444.669.255'
IPADDR='222.444.666.220'
MTU=''
NETMASK='255.255.252.0'
NETWORK='222.444.666.0'
REMOTE_IPADDR=''
STARTMODE='onboot'
UNIQUE='1zs7.+0LdIaFhUB0'
_nm_name='bus-pci-0001:d8:01.0'
BONDING_MASTER=yes
BONDING_MODULE_OPTS='miimon=100 mode=balance-xor'
BONDING_SLAVE0="eth-id-00:11:22:33:44:6A"
BONDING_SLAVE1="eth-id-00:11:22:33:44:6B"
```

Here the slave adapters are identified by MAC address. To map `ethX` device names to physical locations, you can use the `lscfg` command. This is part of the `lsvpd` RPM that ships on CD3 of the SLES9 for POWER™ distribution, but is not installed by default. We recommend that you always install this RPM, which provides a number of hardware-related functions similar to what is known from AIX 5L:

```
sysadmin@SN> rpm -qa|grep lsvpd
lsvpd-0.14.1-1
sysadmin@SN> lscfg|grep eth
+ eth500                U787B.001.DNW5D09-P1-C3-T1
+ eth501                U787B.001.DNW5D09-P1-C3-T2
+ eth502                U787B.001.DNW5D09-P1-C4-T1
+ eth503                U787B.001.DNW5D09-P1-C4-T2
```

```

+ eth504          U787B.001.DNW5D09-P1-T9
+ eth505          U787B.001.DNW5D09-P1-T10
+ eth506         U787B.001.DNW5D09-P1-C2-T1
+ eth507         U787B.001.DNW5D09-P1-C2-T2

```

**Note:** We have observed that after upgrades to the Linux kernel, **lscfg** sometimes reports the device ethX as eth50X, as in the output above. This is an artifact of the **lscfg** command only; the device is still ethX. This bug can be fixed by either manually subtracting 500 from the **lscfg** output, or by reapplying the latest patch level of the lsvpd RPM.

We want to use the two ports of the adapter in location U787B.001.DNW5D09-P1-C2. The MAC address of a device ethX can be found in the **ifconfig ethX** output, but *only* if it is not already enslaved in a bonding device:

```

sysadmin@SN:/etc/sysconfig/network> ifconfig|grep "eth[67]"
eth6      Link encap:Ethernet  HWaddr 00:11:22:33:44:6A
eth7      Link encap:Ethernet  HWaddr 00:11:22:33:44:6B

```

As a reminder that the individual Ethernet devices are enslaved in a channel, we create placeholder configuration files for them in `/etc/sysconfig/network/`, as follows:

```

sysadmin@SN:/etc/sysconfig/network> cat ifcfg-eth-id-00:11:22:33:44:6A
BOOTPROTO='none'
STARTMODE='off'

```

```

sysadmin@SN:/etc/sysconfig/network> cat ifcfg-eth-id-00:11:22:33:44:6B
BOOTPROTO='none'
STARTMODE='off'

```

Looking at the HWaddr of the interfaces in **ifconfig** *after* the channel has been brought up, you can see that the bond0 channel has enslaved eth6 and eth7, as they have the same MAC address as the bonding device. By default, a bonding device inherits its MAC address from the first slave, and sets this on all slaves:

```

sysadmin@SN:/etc/sysconfig/network> ifconfig|grep"HWaddr"
bond0    Link encap:Ethernet  HWaddr 00:11:22:33:44:6A
eth0      Link encap:Ethernet  HWaddr 00:11:22:33:44:6C
eth1      Link encap:Ethernet  HWaddr 00:11:22:33:44:6D
eth2      Link encap:Ethernet  HWaddr 00:11:22:33:44:7A
eth3      Link encap:Ethernet  HWaddr 00:11:22:33:44:7B
eth5      Link encap:Ethernet  HWaddr 00:11:66:44:33:7F
eth6     Link encap:Ethernet  HWaddr 00:11:22:33:44:6A
eth7     Link encap:Ethernet  HWaddr 00:11:22:33:44:6A

```

**Attention:** At our software level, the SLES9 configuration tool **yast** could not handle channel bonding and even corrupted already existing configurations. We recommend that you do *not* use these tools for network interface configuration if you use channel bonding, but rather perform all configuration through the configuration files in `/etc/sysconfig/network/`.

## EtherChannel on Cisco

We do not cover Cisco configuration in detail, but only provide references to some of the commands used to configure the EtherChannels:

- ▶ Be aware that Cisco supports two EtherChannel protocols: *Port Aggregation Protocol* (PAgP), which is a Cisco-proprietary protocol and can only be used across Cisco

switches, and *Link Aggregation Protocol* (LACP), which is the standard IEEE 802.3ad protocol. Make sure that you use LACP. This can be set on a per linecard (“module”) basis by the `set channelprotocol lacp` command.

- ▶ You can set the hashing algorithm with the `set port channel all distribution {ip | mac | session | ip-vlan-session} [source | destination | both]` command, with obvious meanings of the attributes (in Cisco terminology, a *session* is the *port* number).
- ▶ Use the `show lacp channel hash` command to manually verify which port is used for a specific combination of source/destination MAC/address/session values.
- ▶ Statistics on the channel can be displayed by the `show lacp-channel traffic` command.

For details on these commands, refer to the Cisco documentation in “References” on page 43.

## Routing setup for cross-cluster connectivity

In a GPFS 2.3 multicluster environment, every GPFS node in each cluster needs TCP/IP connectivity to every GPFS node in all of the clusters. The main reason for this is that the node that first opens a file will become the GPFS metadata manager for that file, and other nodes that want to access that file need to communicate with it. For *multi-homed* nodes (nodes that are connected into more than one network), exactly one interface is used when a GPFS 2.3 cluster is set up, and the above connectivity statement applies to this interface (and its corresponding IP address and IP name).

This implies that all IONs, the SN and FEN need to be able to contact the eHPS interface of each “JUMP” node (with suffix “f”), and those nodes need to be able to communicate with the IONs, SN and FEN on the Blue Gene I/O network (SN and FEN with suffix “b”). The physical connection has been made as described in Figure 1 on page 4. To complete the configuration we need to set up suitable routing.

**Note:** GPFS version 3.1 introduces several new features, including support for multiple *subnets*. This is advantageous for GPFS configurations on multi-homed hosts and GPFS multiclusters.

### Routing on the pSeries/eHPS cluster

On the pSeries side there are two different cases:

- ▶ On the four NSD servers, nothing needs to be done because they have a direct connection into the Blue Gene I/O network. These four nodes will contact GPFS nodes in the Blue Gene cluster through their local interface in the 222.444.666/22 network:

```
222.444.666.0    222.444.666.202  UHsb  0      0  en7 - - - =>
222.444.666/22  222.444.666.202  U      2 44642474 en7 - - -
222.444.669.255 222.444.666.202  UHsb  0      2  en7 - - -
```

- ▶ On all other nodes in the pSeries cluster, we add a route to the 222.444.666/22 network through the DEISA switch/router. On the DEISA Gigabit, that switch/router has the address [111.222.444.1](#) and all pSeries nodes know that they can reach this address through their own local interface on the [111.222.444/24](#) network.

In AIX 5L, adding the network route can be done through `smitty mkroute`, which will invoke `chdev -l inet0` to permanently add the route to the ODM (the `route add` command will not be persistent). Here is the `netstat -rn` output for the 64/22 network:

```
222.444.666/22    111.222.444.1    UGc    0      0  en1 - - -
```

## Routing on the Blue Gene SN and FEN(s)

On the SN and FEN(s) that run SLES9, routing can be set up in the routes file in the `/etc/sysconfig/network/` directory. We set up a network route to the eHPS *network* through gateway 222.444.666.1, which is the DEISA switch/router's leg into the Blue Gene I/O network. To utilize the direct connections to the four NSD servers, we add *host* routes to their eHPS interfaces through their interfaces on the Blue Gene I/O network. Host routes are more specific than network routes and are thus preferred.

```
sysadmin@SN:/etc/sysconfig/network> cat /etc/sysconfig/network/routes
...
111.222.333.0 222.444.666.1 255.255.255.0 -
111.222.333.202 222.444.666.202 255.255.255.255 -
111.222.333.203 222.444.666.203 255.255.255.255 -
111.222.333.205 222.444.666.205 255.255.255.255 -
111.222.333.206 222.444.666.206 255.255.255.255 -
```

After activating these settings, the routes can be verified with the `route` command.

## Routing on the Blue Gene IONs

For the IONs, which are network booted, routing can be set up through an rc-script that is placed into the `$BGL_SITEDISTDIR` directory on the SN (typically `/bgl/dist/`). We add the same routes as on the SN and FEN(s):

```
sysadmin@SN> cat /bgl/dist/etc/rc.d/rc3.d/S14routes_to_federation

#!/bin/sh
# network route to federation network:
/sbin/route add -net 111.222.333.0 netmask 255.255.255.0 gw 222.444.666.1

# host routes to federation interfaces of I/O nodes via their BlueGene
interfaces:
# (specified host routes have preference over above network route)
/sbin/route add -host 111.222.333.202 gw 222.444.666.202 # nsd01f
/sbin/route add -host 111.222.333.203 gw 222.444.666.203 # nsd02f
/sbin/route add -host 111.222.333.205 gw 222.444.666.205 # nsd03f
/sbin/route add -host 111.222.333.206 gw 222.444.666.206 # nsd04f

# show new routing table:
/bin/netstat -rn
```

To verify that it works, check the IONs logfile at `$BGLLOGS/R??-M?-N?-I:J1?-U?1.log` for the output of the `netstat -rn` command. After the SSH setup is complete (see next section), root on the SN could also log in to IONs which are part of currently booted partitions, and verify that the routing has been correctly set up with the `/sbin/route` command.

## Routing on the Cisco switches

There are two Cisco switches involved: the DEISA switch/router and the Blue Gene switch.

### **DEISA switch/router**

The DEISA switch/router, “deisa”, connects the pSeries cluster with the other DEISA sites and the new Blue Gene system. For the purpose of this discussion, it has two interfaces:

- ▶ A router interface `111.222.444.1` on the pSeries cluster's Ethernet (“m” suffix). This address is used as the gateway for all multicluster traffic on the *pSeries* side.
- ▶ The DEISA switch/router also has a leg into the Blue Gene network, with IP address `222.444.666.1` and IP name `deisa-b` (“b” suffix). This address is used as the gateway for

multiclustert traffic on the Blue Gene side (with the exception of the NSD servers, which have direct connectivity to the Blue Gene Ethernet).

Cisco then routes traffic for the Blue Gene network 222.444.666/22 to address 222.444.666.1, which is its own leg into the Blue Gene network, named deisa-b. Packets addressed to the eHPS interfaces of the pSeries cluster (“f” suffix) are routed to that node’s Ethernet interface (“m” suffix) through the 111.222.444.1 interface on the switch/router.

### **Blue Gene switch**

The Blue Gene I/O network is only switched, and does not need any routing setup on the Blue Gene Cisco switch, zam870. The Blue Gene cluster nodes use 222.444.666.1 as their gateway to the pSeries cluster. This is the DEISA switch/router’s leg on the Blue Gene network, and the DEISA switch/router does all the routing as described above.

**Note:** Similar routing needs to be set up for all other DEISA sites, even if they do not share any file systems with the Blue Gene system. The fact that the pSeries cluster shares some file systems with DEISA and some *other* file systems with Blue Gene necessitates this any-to-any connectivity. This configuration has been omitted here, but it is essential for stable operation of the GPFS 2.3 multiclustert. This requirement is relaxed with GPFS 3.1.

## **Other network-related customization**

The final network-related setup step is to verify that several network services that are needed on the IONs (or are at least convenient to have) are correctly set up.

### **TCP/IP tuning**

The usual TCP/IP tunables should be adapted to achieve higher bandwidth. We do this through a new startup-script in \$BGL\_SITEDISTDIR, using the values recommended in the GPFS documentation:

```
sysadmin@SN> cat /bgl/dist/etc/rc.d/rc3.d/S09etc_sysctl_conf

#!/bin/sh
cat <<E_0_F >> /etc/sysctl.conf
#BEGIN tuning for GPFS
net.core.rmem_max = 8388608
net.core.wmem_max = 8388608
net.ipv4.tcp_rmem = 4096 262144 8388608
net.ipv4.tcp_wmem = 4096 262144 8388608
net.core.netdev_max_backlog = 2500
#END tuning for GPFS
E_0_F
sysctl -p
```

By invoking `sysctl -p` we enforce a reload of the configuration to ensure that the updated values in the configuration file `/etc/sysctl.conf` become effective.

### **Network Time Protocol**

In a cluster, synchronizing time is critical. While the Blue Gene system software ships with NTP installed by default, it may be necessary to generate an `/etc/ntp.conf` file that works for your environment. For example:

```
sysadmin@SN> cat /bgl/dist/etc/rc.d/rc3.d/S14etc_ntp_conf

#!/bin/sh
```

```
echo "restrict default nomodify" > /etc/ntp.conf
echo "server 222.444.666.220" >> /etc/ntp.conf
echo "authenticate no" >> /etc/ntp.conf
```

Here the server is the Blue Gene Service Node. If you use another NTP server, you also need to establish routing to that server from the IONs.

## Name resolution

Although all Blue Gene IP addresses and names have been added to `/etc/hosts`, it is useful to have DNS name resolution working on the IONs. Simply create the `/etc/resolv.conf` file (and make sure routes to your DNS servers exist), and name resolution should work as usual:

```
sysadmin@SN> cat /bgl/dist/etc/rc.d/rc3.d/S14etc_resolv_conf

#!/bin/sh
echo "nameserver 111.333.555.2" > /etc/resolv.conf
echo "nameserver 111.333.555.3" >> /etc/resolv.conf
echo "nameserver 111.333.555.4" >> /etc/resolv.conf
echo "search your-domain.org your-domain.org" >> /etc/resolv.conf
```

## Blue Gene CIOD tuning

One final area that needs customization is the CIOD daemon that runs on the IONs. The CIOD daemon is responsible for forwarding of I/O traffic between the compute nodes and the Ethernet interface of their pSet's ION. It creates one memory buffer for each compute node *CPU* (two per *node* to accommodate virtual node mode). The size of this buffer should ideally match the size of the I/O operations. We set it to the file system blocksize of our GPFS file systems, which is 1 MB, and also experimented with twice that size:

```
sysadmin@SN> cat /bgl/dist/etc/rc.d/rc3.d/S10fzj_sysconfig

#!/bin/sh
...
echo "export CIOD_RDWR_BUFFER_SIZE=2097152" >> /etc/sysconfig/ciod
echo "export DEBUG_SOCKET_STARTUP=ALL" >> /etc/sysconfig/ciod
```

This configuration file and the available options are described in the `ionode.README` for the Blue Gene system software.

## SSH setup for the Blue Gene GPFS cluster

All nodes *within* a GPFS cluster must be able to communicate via `rsh` or `ssh`. Under Linux the default is to use `ssh`. This has to be configured for the Blue Gene cluster, so the root user on each GPFS node in that cluster can `ssh` to any other node in that cluster without prompting for a password.

**Note:** There is no requirement for `ssh` access *between* clusters. All inter-cluster communication is handled by the `mmfsd` daemons (which internally use SSL).

There are three different types of public/private key pairs involved:

- ▶ An SSH daemon's key, used to authenticate a *host* in `ssh` connections.
- ▶ A (root) user's personal SSH key, used to authenticate a (root) *user* in `ssh` connections.

- ▶ An SSL key, used to authenticate an *application*. In this case, the application is the `mmfsd` daemon on one GPFS cluster, communicating with the `mmfsd` daemon on another GPFS cluster.

Here we describe the setup of the first two, which can be performed independently from the GPFS software *installation* but must be completed before GPFS *configuration*. The cross-cluster SSL setup for the GPFS daemons is part of the GPFS configuration described in “Creating a single-node Blue Gene GPFS cluster on the service node” on page 22. The following steps are needed, separately for the SN, the FEN(s), and the IONs:

1. The SSH software needs to be installed.
2. The SSH daemon’s keys must be created to *authenticate* the host.
3. The root user keys must be created to *authenticate* the root user.
4. The SSH daemon needs to be started.
5. The SSH daemons’ public keys must be made known to all other cluster nodes to avoid interactive prompts when an `ssh` client connects to a remote `sshd` for the first time.
6. The authorization files must be configured to *authorize* the root user on one cluster node for `ssh` access to the root user of another cluster node (to avoid interactive password prompts).

Because the detailed process is slightly different for the SN/FEN(s) and Blue Gene ION(s), we discuss steps 1 to 4 separately, and then discuss steps 5 and 6.

## SSH setup on the SN and FEN(s)

Repeat the following separately on the SN and each FEN:

1. SSH is installed by default when a SLES9 server is installed.
2. The SSH daemon’s key-pair is created automatically during installation.
3. To create a key-pair for the root user, run the `ssh-keygen` command. We explicitly set a comment field to indicate the key’s usage (“-C” option, change text as appropriate). No passphrase is allowed (“-N” option) because the key will be used non-interactively:

```
root@SN> ssh-keygen -t dsa -b 1024 -f ~/.ssh/id_dsa -N '' -C
'root@bluegene-sn'
```

4. SSH is started automatically on SLES9.

## SSH setup for the IONs

Because the IONs do not have local disks, the SSH setup for them is slightly different:

1. The SSH software is already installed in the `$BGOS` tree that is shipped with the Blue Gene system software. This directory is mounted by all IONs during ION bring-up:

```
export BGL0S=/bgl/BlueLight/ppcfloor/bgl/sys/bin/bgl0S
rpm --root $BGL0S -qa | grep -i -E "ssh|ssl"
# openssh-3.8p1-48
# openssl-0.9.7d-24
```

Configuration information (like key-pairs) is specific for each customer installation, so all the SSH configuration for the IONs takes place in the `$BGL_SITEDISTDIR` directory on the SN, normally `/bgl/dist/`. This directory is also mounted by the IONs during ION bring-up.



2. We habitually create both RSA and DSA keys (“-t” option), although one key type would be sufficient. No passphrase is allowed to avoid interactive prompts (“-N” option). We provide a description of the keys with the “-C” option:

```
mkdir -p /bgl/dist/etc/ssh
chmod 755 /bgl /bgl/dist /bgl/dist/etc /bgl/dist/etc/ssh
cd /bgl/dist/etc/ssh
ssh-keygen -t rsa -b 1024 -f ./ssh_host_rsa_key -N '' -C 'bluegene-ion'
ssh-keygen -t dsa -b 1024 -f ./ssh_host_dsa_key -N '' -C 'bluegene-ion'
ls -al /bgl/dist/etc/ssh
```

3. The root user key-pair for the IONs also has to be created without a passphrase:

```
mkdir -p /bgl/dist/root/.ssh
chmod 700 /bgl/dist/root /bgl/dist/root/.ssh
ls -ld /bgl/dist/root /bgl/dist/root/.ssh
#drwx----- 3 root root 72 2006-02-15 15:29 /bgl/dist/root
#drwx----- 2 root root 48 2006-02-15 15:29 /bgl/dist/root/.ssh

ssh-keygen -t dsa -b 1024 -f /bgl/dist/root/.ssh/id_dsa -N '' -C
'root@bluegene-ion'
ls -al /bgl/dist/root/.ssh
```

Note that we treat all IONs as identical and use a single key-pair for *all* IONs, and a single key-pair for *all* root users on these IONs. This should be OK and makes the setup much easier than generating different keys for the different IONs and users.

4. Startup of the SSH daemon is controlled by the S16sshd rc-script in the \$BGL\_DISTDIR directory, normally /bgl/BlueLight/ppcfloor/dist/. This file is always invoked at ION startup, but will only start SSH when any of the following two conditions is met:
  - GPFS will be started (see below how this is configured).
  - The /etc/sysconfig/sshd file is present locally on the ION (arbitrary contents).

If you set up GPFS correctly, this will automatically start up the SSH daemons and no further action is necessary. However, we prefer to verify that SSH works correctly *before* starting the GPFS configuration. So we use the second option to start up SSH before we proceed with GPFS. Within our site-specific rc-file /bgl/dist/etc/rc.d/rc3.d/S10fzj\_sysconfig, we create the startup file as follows:

```
echo "# if an /etc/sysconfig/ssh file is present," >
/etc/sysconfig/ssh
echo "# the driver's S16sshd starts SSHD even w/o GPFS..." >>
/etc/sysconfig/ssh
```

The next Blue Gene block that will be booted will start SSH on the IONs of that block, through the /bgl/BlueLight/ppcfloor/dist/etc/rc.d/rc3.d/S16sshd script. Make sure that you use a sequence number below 16 for your site-specific script, to ensure that you create the file /etc/sysconfig/sshd before the S16sshd script checks for its presence.

## SSH public key exchanges within the cluster

After the key-pairs for all cluster nodes and for the root users on all cluster nodes have been *generated*, these keys must be *distributed* to all cluster nodes: the SSH daemons’ public keys are collected in a known\_hosts file, and the root user public keys are added to the root users’ authorized\_keys files to authorize **ssh** connections without a password.

5. Making the SSH daemons’ public keys known

Whenever an **ssh** client connects to a remote **sshd** server, it verifies the remote server’s authenticity by using a copy of that SSH daemon’s public key that is stored in the client’s

list of *known hosts*. If no local copy of the SSH daemon's public key exists, an interactive prompt asks the user if the public key should be added to the list of *known hosts*. GPFS will use **ssh** non-interactively, so this interactive prompt *must* be prevented for connections within the GPFS cluster. This can be achieved by manually adding the affected public keys to the *known hosts* file, on all nodes in the cluster, before starting up GPFS. The public keys of other cluster nodes can be stored in two places:

- The global `/etc/ssh/ssh_known_hosts` file
- The per-user `$HOME/.ssh/known_hosts` file

We prefer the system-wide file because it is easier to manage, and applies to all users. Create the file on the SN, and then copy it to the FEN(s) and to a location in `/bgl/` where the IONs can access it. For the IONs we use only a single key-pair, but their IP names and addresses are different, of course. SSH allows some wildcarding (see `man ssh`) so we do not need to add all IONs individually. Our `/etc/ssh/ssh_known_hosts` file contains the following:

```
cat /etc/ssh/ssh_known_hosts

# Service Node (Blue Gene functional network IP, and admin LAN IP):
SN,SN.your-domain.org,111.333.555.111,SN-b,SN-b.your-domain.org,222.444.666.
220 ssh-rsa AAAABBBBCCCC.....KaA8E=
# Frontend Node (Blue Gene functional network IP, and user LAN IP):
FEN,FEN.your-domain.org,111.333.555.10,FEN-b,FEN-b.your-domain.org,222.444.6
66.221 ssh-rsa AAAABBBBCCCC.....1jJFc=
# IONs (short and long IP names, using SSH config file wildcards):
R??-M?-N?-?,R??-M?-N?-?.your-domain.org ssh-rsa AAAABBBBCCCC.....jDDOk=
# IONs (IP addresses, using SSH config file wildcards where possible):
222.444.666.33,... ssh-rsa AAAABBBBCCCC.....jDDOk= root@bluegene-ion
.....
```

#### Notes:

We recommend to add the IP address, short IP name, and long IP names of the SN, FEN(s) and IONs to the known hosts file. This ensures that **ssh** will work without interactive prompts regardless of how the remote host is addressed. For the SN and FEN we also include their interfaces of the external network connections. This is not necessary for GPFS, but convenient for users who are using **ssh** for connections on those networks.

For the IONs' IP addresses, make sure that you do not use wildcards that match a bigger pattern than the IP range that the IONs actually use (for example, the SN, FEN, or file server IP addresses that are in the same network). Such "duplicate public key matches" may lead to undesirable effects.

After the file is completed on the SN, we make sure it is world-readable and copy it to the FEN(s). For the IONs, we copy the file both to the system-wide file and to the root user's personal file. The Blue Gene V1R2 system software bring-up procedure honors the root user's personal file, but not the system-wide file:

```
chmod 644 /etc/ssh/ssh_known_hosts

scp -p /etc/ssh/ssh_known_hosts root@FEN:/etc/ssh/ssh_known_hosts

cp -p /etc/ssh/ssh_known_hosts /bgl/dist/etc/ssh/ssh_known_hosts
# this does NOT get pulled in on IONs
cp -p /etc/ssh/ssh_known_hosts /bgl/dist/root/.ssh/known_hosts
```

```
# this DOES get pulled in on IONs
```

## 6. Setting up ssh authorization

The final step in the SSH setup is to *authorize* the root user to run **ssh** connections across all cluster nodes without a password. This is done by creating an `authorized_keys` file that contains the SN, FEN, and ION root users' public keys that have been created in step 3 above:

```
# add root@SN pubkey:
cat ~root/.ssh/id_dsa.pub          >> ~root/.ssh/authorized_keys

# get root@FEN pubkey and add it:
scp root@FEN:~/.ssh/id_dsa.pub \
  ~root/.ssh/FEN-id_dsa.pub # this will still ask for a password ;-)
cat ~root/.ssh/FEN-id_dsa.pub     >> ~root/.ssh/authorized_keys

# add root@IONs pubkey:
cat /bgl/dist/root/.ssh/id_dsa.pub >> ~root/.ssh/authorized_keys

# copy the file to the root user's .ssh directory on the IONs:
# (we also have some administrators' personal pubkeys in this file on the
# SN,
# hence the grep for the root@bluegene* users keys' comment fields set
# above...)
grep -E "root@bluegene" ~root/.ssh/authorized_keys \
  >> /bgl/dist/root/.ssh/authorized_keys

# copy the file to the root user's .ssh directory on the FEN:
scp /bgl/dist/root/.ssh/authorized_keys \
  root@FEN:~/.ssh/authorized_keys # this will still ask for a password
;-)
```

As for the known hosts file, we first created the authorization file on the SN. Then we copied it to a location in `/bgl/` where the IONs can access it, and also to the FEN(s).

## Testing the SSH connectivity

Now the SSH setup is complete and can be tested. Allocating a block (partition) in the Blue Gene racks will start up SSH on its IONs, and **ssh** connections between the SN, FEN(s) and these IONs can then be tested. For example, from the SN:

```
# local script to allocate a block to a user
bglallocate R000_N0123 someuser # a 128-way block with 4 node cards and 4
IONs

ssh 222.444.666.34 uptime          # test IP addresses
ssh r00-m0-n0-4 uptime            # test short IP names
ssh r00-m0-n0-4.your-domain.org uptime # test long IP names
```

After this verification step, we know that SSH works and can now install and configure GPFS.

## GPFS installation and configuration

As described in the *GPFS HOWTO for Blue Gene*, the Blue Gene GPFS cluster consists of the SN, FEN(s), and IONs. It does not own any GPFS file systems, but accesses the remote GPFS cluster's file systems. To set up this cluster we proceed as follows:

- ▶ “GPFS software installation” on page 20.
- ▶ “Creating a single-node Blue Gene GPFS cluster on the service node” on page 22.
- ▶ “Establishing the GPFS multicluster access” on page 24.
- ▶ “Adding the FEN(s) to the Blue Gene GPFS cluster” on page 28.
- ▶ “Adding the IONs to the Blue Gene GPFS cluster” on page 29.
- ▶ “Customizing the GPFS startup process for the IONs” on page 32.

The general planning and installation steps to set up a GPFS cluster are described in *GPFS 2.3 Concepts, Planning and Installation Guide*, GA22-7968-02.

## GPFS software installation

For this study the Blue Gene system software was at the V1R2M1 level. The minimum prerequisite level for GPFS 2.3 on Blue Gene is V1R2M0, but we recommend to upgrade to the latest available Blue Gene software level before starting to install GPFS.

**Note:** The Blue Gene system software version V1R3 requires GPFS 3.1.

### Obtaining the required GPFS software

There are three different sets of hardware/OS involved in this study, and correspondingly three different pieces of GPFS software:

1. The “OWNING” pSeries cluster consists of 41 frames of p690 servers, all running AIX 5L, maintenance level 5200-06, and GPFS 2.3 for AIX 5L. This cluster is at GPFS version 2.3.0.9 (with base level for AIX 5L from CD, and PTF levels from the AIX 5L fix distribution center or from:

<http://techsupport.services.ibm.com/server/gpfs/download/home.html>)

We do not discuss this further because this cluster is already operating (this is standard GPFS for AIX 5L).

2. The Blue Gene service node (hostname SN) and front-end node (hostname FEN) are two POWER5-based OpenPower 720 servers running SLES9 SP3 for pSeries. They need GPFS 2.3 for SLES9 on pSeries. At the time of installation, patch levels 2.3.0.7 to 2.3.0.10 were available, with base level for SLES from CD, and PTF levels from:

<http://techsupport.services.ibm.com/server/gpfs/download/home.html>

**Tip:** GPFS requires that you accept its license before installation. With AIX 5L this can be done via `installp`, but the Linux `rpm` command has no “license acceptance” option. Instead, the RPM files are shipped in a “locked” format. After copying the CD contents to disk you need to run the following to accept the GPFS license under Linux. This will unpack the individual base level RPM files:

```
./gpfs_install-2.3.0-0_sles9_ppc64 --dir . --text-only --silent
```

3. The Blue Gene I/O nodes (IONs) are PPC440-based Blue Gene processors and run an embedded Linux kernel. A GPFS 2.3 version for Blue Gene is available as PRPQ p91224. At the start of this study only PTF level 2.3.0.8 was available; later on updates including 2.3.0.13 became available. For customers that have ordered the PRPQ, the software is available from the Blue Gene software download site (installable PTF levels, no need for a base level CD).

Although it may be possible to have mixed PTF levels within a cluster, we decided to start with a homogeneous setup and use 2.3.0.8 for all participants in the Blue Gene cluster (SN, FEN, and IONs). Later in the process we upgraded all participants to 2.3.0.13.

## Installing the GPFS software on the SN and FEN(s)

This is a standard GPFS for pSeries Linux installation. Before installing, consult the *GPFS FAQ* for the minimum Linux kernel level required for your level of GPFS. Verify which kernel level you are running (for example, by `uname -r`), and upgrade if you are downlevel.

You have to perform the following actions on the SN and each FEN in your Blue Gene system. For larger clusters, you may want to use a cluster management software such as CSM. Here we have only one SN and one FEN, and install the software manually.

1. Install the GPFS base level and PTF level RPMs:

```
cd /install/gpfs-2.3.0-0 ; rpm -ivh gpfs*.rpm
cd /install/gpfs-2.3.0-8 ; rpm -Uvh gpfs*.rpm
rpm -qa | grep gpfs # verify that the install worked...
```

2. Build the GPFS Portability Layer (GPL) as described in `/usr/lpp/mmfs/src/README`. Note that you need the Linux kernel sources installed at the level matching your running kernel, as well as the gcc compiler and some other prerequisite RPMs.

```
export SHARKCLONEROOT=/usr/lpp/mmfs/src

cd /usr/lpp/mmfs/src/config
cp site.mcr.proto site.mcr
vi site.mcr
diff site.mcr*|grep "^<"
#< #define GPFS_ARCH_PPC64
#< LINUX_DISTRIBUTION = SUSE_LINUX
#< #define LINUX_KERNEL_VERSION 2060507
#< KERNEL_HEADER_DIR = /lib/modules/`uname -r`/source/include

cd /usr/lpp/mmfs/src
make World 2>&1 | tee /tmp/make-world.txt
make InstallImages 2>&1 | tee /tmp/make-installimages.txt

cd /usr/lpp/mmfs/bin
ls -al mmfslinux mmfs26 lxtrace tracedev dumpconv
#-r-x----- 1 root root 44655 2006-02-15 12:20 tracedev
#-r-x----- 1 root root 15667 2006-02-15 12:20 dumpconv
#-r-x----- 1 root root 42678 2006-02-15 12:20 lxtrace
#-r-x----- 1 root root 1649984 2006-02-15 12:20 mmfs26
#-r-x----- 1 root root 294632 2006-02-15 12:20 mmfslinux
```

## Installing the GPFS software for the IONs

The Blue Gene nodes are diskless. The IONs are booted over the network and then mount the `$BGL0S` directory from the SN. The data in that tree originally comes with the Blue Gene system software. The (optional) GPFS software can be installed into that tree by changing the RPM base directory with the `--root` option:

```
export BGL0S=/bg1/BlueLight/ppcfloor/bglsys/bin/bg10S
```

```
rpm --root $BGL0S -qa | grep -E 'ssl|ssh' # see which RPMs are installed,
need SSH...
```

```
cd /install/gpfs-bgl-prpq-2.3.0-8
rpm --root $BGLOS --nodeps -iv \
  gpfs.base-2.3.0-8.ppc.rpm \
  gpfs.docs-2.3.0-8.noarch.rpm \
  gpfs.gplbin-2.3.0-8.ppc.rpm \
  gpfs.msg.en_US-2.3.0-8.noarch.rpm
```

```
rpm --root $BGLOS -qa | grep gpfs # verify that the install worked...
```

**Attention:** Updates to the Blue Gene system software (like an update from V1R2M0 to V1R2M1) will replace the \$BGLOS directory. So after each such update you need to reinstall the GPFS software into \$BGLOS.

Note that a gpfs.gplbin file set is installed, which does not exist for other GPFS Linux versions. This is a precompiled version of the GPFS Portability Layer (GPL) executables, so you do not need to build the portability layer manually as above for the SLES9 install. You can check that the GPL executables are present by looking into the GPFS binary directory in \$BGLOS:

```
cd $BGLOS/usr/lpp/mmfs/bin
ll mmfslinux mmfs26 ltrace tracedev dumpconv
#-r-x----- 1 root root 14365 2005-11-17 20:49 dumpconv
#-r-x----- 1 root root 34930 2005-11-17 20:49 ltrace
#-r-x----- 1 root root 190758 2005-11-17 20:49 mmfslinux
#-r-x----- 1 root root 16????? 2005-11-17 20:49 mmfs26
#-r-x----- 1 root root 16093 2005-11-17 20:49 tracedev
```

Alternatively you can verify this by checking the contents of the gpfs.gplbin RPM file with the `rpm -qlp gpfs.gplbin-2.3.0-8.ppc.rpm` command.

Before starting to configure GPFS, it is worthwhile to add the GPFS binary directory to root's \$PATH on the SN and FEN(s):

```
echo 'export PATH=/usr/lpp/mmfs/bin:$PATH' >> ~root/.profile
. ~root/.profile
```

This enables root to run the GPFS commands without specifying their full pathnames.

## Creating a single-node Blue Gene GPFS cluster on the service node

First we create a GPFS cluster on the SN using `mmcrcluster`. The SN will be the primary *cluster configuration server* and no backup configuration server will be configured (at a later time we may add the FEN as a secondary). We verify the cluster creation with `mm1scluster` and `mm1sconfig`:

**Note:** Normally it is not advisable to have only a single quorum node and cluster configuration server because this would introduce a single point of failure. However, in the Blue Gene environment the Service Node needs to be available for the Blue Gene IONs to be usable in the first place. So a GPFS cluster with the SN as quorum node and IONs dynamically joining and leaving the cluster as non-quorum nodes does not create an additional availability risk.

```
echo 'SN-b:manager-quorum' > /tmp/gpfs_nodfile.sn

mmcrcluster \
  -n /tmp/gpfs_nodfile.sn \
```

```

-p SN-b.your-domain.org \
-r /usr/bin/ssh \
-R /usr/bin/scp \
-C CLIENT.your-domain.org \
-U CLIENT.your-domain.org \
-A
#Wed Feb 15 16:50:05 CET 2006: mmcrcluster: Processing node
SN-b.your-domain.org
#mmcrcluster: Command successfully completed

```

**mm1scluster**

```

#
#GPFS cluster information
#=====
# GPFS cluster name:      CLIENT.your-domain.org
# GPFS cluster id:       987654321987654321
# GPFS UID domain:       CLIENT.your-domain.org
# Remote shell command:  /usr/bin/ssh
# Remote file copy command: /usr/bin/scp
#
#GPFS cluster configuration servers:
#-----
# Primary server:   SN-b.your-domain.org
# Secondary server: (none)
#
# Node number  Node name      IP address      Full node name      Remarks
#-----
#           1      SN-b           222.444.666.220    SN-b.your-domain.org  quorum node

```

**mm1sconfig**

```

#Configuration data for cluster CLIENT.your-domain.org:
#-----
#clusterName CLIENT.your-domain.org
#clusterId 987654321987654321
#clusterType lc
#multinode yes
#autoload yes
#useDiskLease yes
#uidDomain CLIENT.your-domain.org
#maxFeatureLevelAllowed 813
#
#File systems in cluster CLIENT.your-domain.org:
#-----
#(none)

```

Two important settings that typically need to be adjusted are the maximum blocksize and the size of the GPFS pagepool. Beware that the pagepool *must not* exceed 50% of main memory, which on the IONs normally is 512 MB, so we are using 128 MB as a conservative starting point. The settings can be changed with the **mmchconfig** command, and when we then run **mm1sconfig** again it will show the changed values:

```

mmchconfig maxblocksize=1024K
#mmchconfig: Command successfully completed
mmchconfig pagepool=128M
#mmchconfig: Command successfully completed

```

```

mmfsconfig
#Configuration data for cluster CLIENT.your-domain.org:
#-----
#...
#maxblocksize 1024K
#pagepool 128M
#...

```

**Note:** Values like pagepool can be set either globally as above, or for a subset of nodes. For example, this allows different settings on the IONs and the external System p™ servers.

To verify that the newly created GPFS cluster works correctly, we start and stop it:

```

mmstartup -a
#Wed Feb 15 16:57:08 CET 2006: mmstartup: Starting GPFS ...

mmshutdown -a
#Wed Feb 15 17:11:18 CET 2006: mmshutdown: Starting force unmount of GPFS file
systems
#Wed Feb 15 17:11:23 CET 2006: mmshutdown: Shutting down GPFS daemons
#SN-b.your-domain.org: Shutting down!
#SN-b.your-domain.org: 'shutdown' command about to kill process 2319
#SN-b.your-domain.org: Unloading modules from /usr/lpp/mmfs/bin
#SN-b.your-domain.org: Unloading module mmfs
#SN-b.your-domain.org: Unloading module mmfslinux
#SN-b.your-domain.org: Unloading module tracedev
#Wed Feb 15 17:11:33 CET 2006: mmshutdown: Finished

```

## Establishing the GPFS multicluster access

With this minimal Blue Gene GPFS cluster, we can now establish the connection to the owning cluster, and access its file systems. We break this down into six steps:

1. “SSL setup for the local CLIENT cluster” on page 24.
2. “SSL setup for the remote OWNING cluster” on page 25.
3. “Key exchange between the two GPFS clusters OWNING and CLIENT” on page 26.
4. “On the remote OWNING cluster, allow access by the CLIENT cluster” on page 26.
5. “On the local CLIENT cluster, access the remote GPFS file systems” on page 27.
6. “On the local CLIENT cluster, mount the remote GPFS file systems” on page 28.

When the multicluster setup works, we can then add the FEN(s) and IONs to the cluster. GPFS will automatically manage the replication of the multicluster setup to the new nodes.

### SSL setup for the local CLIENT cluster

Using the `mmauth` command, we generate an SSL key-pair for the local cluster that can then be used in cross-cluster authentication. That key will be stored in the `/var/mmfs/ssl/` directory. After the key has been created, the cluster configuration can be updated to *use* SSL for authentication, using the `mmchconfig cipherList` command. Note that the latter will fail if no SSL key-pair has been created yet.

```

ll /var/mmfs/ssl/id_rsa.pub # none

# test - should fail...

```



### mmchconfig cipherList=AUTHONLY

```
#mmchconfig: You must first generate an authentication key file. Run: mmauth genkey
```

```
mmauth genkey # needs to be done at least once before cipherList is set
```

```
#Verifying GPFS is stopped on all nodes ...
#Generating RSA private key, 512 bit long modulus
#.....+++++++
#...+++++++
#e is 54321 (0x10101)
#mmauth: Command successfully completed
```

```
ll /var/mmfs/ssl/
#total 16
#drwx----- 3 root root 216 2006-02-15 17:43 .
#drwxr-xr-x 7 root root 168 2006-02-15 11:14 ..
#lrwxrwxrwx 1 root root 7 2006-02-15 17:43 id_rsa -> id_rsa1
#-rw----- 1 root root 493 2006-02-15 17:43 id_rsa1
#-rw-r--r-- 1 root root 457 2006-02-15 17:43 id_rsa.cert
#-rw-r--r-- 1 root root 182 2006-02-15 17:43 id_rsa.pub
#-rw-r--r-- 1 root root 190 2006-02-15 17:43 openssl.conf
#drwx----- 2 root root 48 2006-02-15 16:09 stage
```

```
mmauth show
#Cluster name: CLIENT.your-domain.org (this cluster)
#Cipher list: (none specified)
#SHA digest: abcefd0123456789abcdef0123456789
#File system access: (all rw)
```

### mmchconfig cipherList=AUTHONLY

```
#Verifying GPFS is stopped on all nodes ...
#mmchconfig: Command successfully completed
```

```
mmauth show
#Cluster name: CLIENT.your-domain.org (this cluster)
#Cipher list: AUTHONLY
#SHA digest: abcefd0123456789abcdef0123456789
#File system access: (all rw)
```

```
mmfsconfig|grep cipher
#cipherList AUTHONLY
```

**Note:** With GPFS 3.1, the cipherList setting can alternatively be changed through a new **mmauth** subcommand as well as through the GPFS 2.3 **mmchconfig cipherList** subcommand.

## SSL setup for the remote OWNING cluster

The remote cluster that owns the file systems is already set up for GPFS multicluster. We just verify that everything is in place:

```
ssh nsd01 # any node in the remote cluster

ls /var/mmfs/ssl
authorized_keys          known_cluster.ibmcs.csc.fi
id_rsa                   known_cluster.j39d1m
```

```

id_rsa.cert                known_cluster.psi.rzg.mpg.de
id_rsa.pub                 known_cluster.spmsw102
id_rsa2                    known_cluster.zam452b
known_cluster.IDRPROD.idris.fr  openssl.conf
known_cluster.IDRTEST.idris.fr  stage

```

```

mmfsconfig|grep cipher
#cipherList AUTHONLY

```

An SSL key-pair (`id_rsa` and `id_rsa.pub`) for this cluster is already created and activated. The `authorized_keys` file contains all the public keys of other clusters that are allowed to access file systems of this cluster, and the `known_cluster.clustername` files are the public keys of other clusters whose file systems are accessed from this cluster. These files are managed by the GPFS commands and should not be edited manually.

### Key exchange between the two GPFS clusters OWNING and CLIENT

Now that both clusters have their SSL key-pairs generated and activated, their public keys need to be exchanged. This is a manual file copy process, for example:

```

scp root@SN:/var/mmfs/ssl/id_rsa.pub root@NSD1:/tmp/gpfs-CLIENT.pub
scp root@NSD1:/var/mmfs/ssl/id_rsa.pub root@SN:/tmp/gpfs-OWNING.pub

```

You can use one node in each cluster for this operation. There is no need to copy these files to all nodes in a GPFS cluster. The files are copied into `/tmp` to emphasize that they are used only *once*, as input to the following GPFS commands.

### On the remote OWNING cluster, allow access by the CLIENT cluster

On the OWNING cluster that owns the file systems, we first establish the *authentication* of the CLIENT cluster by adding its public key to the data in `/var/mmfs/ssl/` using the `mmauth add` command. Run the following on the same node to which you copied the public key:

```

mmauth add CLIENT.your-domain.org -k /tmp/gpfs-CLIENT.pub
#mmauth: Command successfully completed
#mmauth: 6027-1371 Propagating the changes to all affected nodes.
#This is an asynchronous process.

mmauth show CLIENT.your-domain.org
#Cluster name:      CLIENT.your-domain.org
#Cipher list:      AUTHONLY
#SHA digest:       abcefd0123456789abcdef0123456789
#File system access: (none authorized)

```

As indicated by the 6027-1371 message, GPFS will manage the distribution of this information to all nodes in the cluster: The whole `/var/mmfs/ssl/` directory is synchronized across all nodes by the GPFS software. The file in `/tmp` is now no longer needed.

At the end of the `mmauth list` command output, it can be seen that this step did not yet establish any *authorization* to access file systems owned by OWNING. To do this we run the `mmauth grant` command for each GPFS file system that we want to make available:

```

for SHAREDIFS in home1 home2 home3 home4 home5 home6 home7 work
do
    echo "granting access for $SHAREDIFS..."
    mmauth grant CLIENT.your-domain.org -f $SHAREDIFS -a rw -r no
done
#
mmauth show CLIENT.your-domain.org

```

```

#Cluster name:      CLIENT.your-domain.org
#Cipher list:       AUTHONLY
#SHA digest:        abcefd0123456789abcdef0123456789
#File system access: home1      (rw, root allowed)
#                  home2      (rw, root allowed)
#...
#                  work       (rw, root allowed)

```

Now the CLIENT cluster is granted access to these GPFS file systems owned by OWNING.

### On the local CLIENT cluster, access the remote GPFS file systems

Analogous to the steps above, we first establish the *authentication* of the OWNING cluster by adding its public key to the data in `/var/mmfs/ssl/`. This time (on the CLIENT cluster), we need to use the `mmremotecluster add` command. Run the following on the same node where you copied the public key into `/tmp`.

In addition to the remote cluster's SSL public key, we also need a list of *contact nodes* that this cluster will use to connect to the remote cluster. By convention we use the *quorum nodes* in the remote cluster. Although this is not a requirement, it is a sensible choice because these nodes are more likely to be available than the non-quorum nodes.

```

ls /var/mmfs/ssl
# .  authorized_keys  id_rsa1      id_rsa.pub   stage
# .. id_rsa           id_rsa.cert  openssl.conf

ssh nsd01 'mmlscluster | grep -E "server:|quorum"'
# Primary server:  nsd01f
# Secondary server: nsd03f
#   37   nsd01f      111.222.333.202  nsd01f      quorum
node
#   38   nsd02f      111.222.333.203  nsd02f      quorum
node
#   41   nsd03f      111.222.333.205  nsd03f      quorum
node
#   42   nsd04f      111.222.333.206  nsd04f      quorum
node

```

```

mmremotecluster show all
#mmremotecluster: There are no remote cluster definitions.

```

```

mmremotecluster add OWNING.your-domain.org \
  -k /tmp/gpfs-OWNING.pub -n nsd01f,nsd02f,nsd03f,nsd04f
# quietly returns with no stdout...

```

```

mmremotecluster show all
#Cluster name:      OWNING.your-domain.org
#Contact nodes:     nsd01f,nsd02f,nsd03f,nsd04f
#SHA digest:        123456789abcefd0123456789abcdef0
#File systems:      (none defined)

```

```

ls /var/mmfs/ssl
# .  id_rsa      id_rsa.pub   stage
# .. id_rsa1    known_cluster.OWNING.your-domain.org
#authorized_keys id_rsa.cert  openssl.conf

```

Now the `known_cluster.clustername` file for the OWNING cluster has been added to the `/var/mmfs/ssl/` directory, but no remote file systems are defined yet. To complete the setup, we need to add the remote file systems using **mmremotefs add**:

```
mmremotefs show all
#mmremotefs: There are no remote file systems.

for SHAREDIFS in home1 home2 home3 home4 home5 home6 home7 work
do
    echo "adding $SHAREDIFS..."
    mmremotefs add $SHAREDIFS -f $SHAREDIFS -C OWNING.your-domain.org -T /$SHAREDIFS
-A no
done

mmremotefs show -C OWNING.your-domain.org
#Local Name Remote Name Cluster name Mount Point Mount Options
Automount
#home1 home1 OWNING.your-domain.org /home1 rw
no
#home2 home2 OWNING.your-domain.org /home2 rw
no
...
#work work OWNING.your-domain.org /work rw
no
```

Note that we did not enable automatic mounting for the remote file systems (option “-A no”) because we first want to test the setup manually. This can be changed later.

### On the local CLIENT cluster, mount the remote GPFS file systems

When we now start GPFS on the SN, all the added remote file systems should be automatically added to `/etc/fstab`, but will not yet be mounted:

```
mmstartup

grep gpfs /etc/fstab
#/dev/home1 /home1 gpfs
rw,dev=OWNING.your-domain.org:home1,ldev=home1,noauto 0 0
#/dev/home2 /home2 gpfs
rw,dev=OWNING.your-domain.org:home2,ldev=home2,noauto 0 0
...
#/dev/work /work gpfs
rw,dev=OWNING.your-domain.org:work,ldev=work,noauto 0 0
```

These entries are correct, so we can mount the file systems, for example `mount /home1`.

## Adding the FEN(s) to the Blue Gene GPFS cluster

The FEN(s) can be added by the **mmaddnode** command. This will also synchronize the multicluster setup, in particular the `/var/mmfs/ssl/` directory, to the added node. We perform all of these actions on the SN.

```
ping FEN-b # to check that the node is reachable...
ssh FEN-b "rpm -qa|grep gpfs.base" # to check that ssh to the node works...

mmaddnode FEN-b:client-nonquorum
#Wed Feb 15 18:15:26 CET 2006: mmaddnode: Processing node FEN-b.your-domain.org
#mmaddnode: Command successfully completed
```

```
#mmaddnode: Propagating the changes to all affected nodes.
#This is an asynchronous process.
```

```
mm1snode -a
#GPFS nodeset      Node list
#-----
# CLIENT           SN-b FEN-b

mm1scluster | grep FEN-b
#      2           FEN-b           222.444.666.221       FEN-b.your-domain.org
```

Now GPFS can be started on the FEN, and the file systems can be mounted on the FEN. Because the FEN is a “normal” server running SLES9, this works exactly like on the SN.

## Adding the IONs to the Blue Gene GPFS cluster

The Blue Gene IONs are different from the SN and FEN(s) in that they are diskless and all their configuration information and startup scripts reside in the /bgl/ directory on the SN. The GPFS-specific information that is normally stored in /var/ is placed in per-ION directories /bgl/gpfsvar/<ion-ip-address>/var/, which are mounted over the ION’s /var/ subdirectories. The /bgl/gpfsvar/ directory needs to be created before adding IONs. The per-ION subdirectories are created by the IONs’ GPFS startup scripts:

```
mkdir          /bgl/gpfsvar
chown root.root /bgl/gpfsvar
chmod 750      /bgl/gpfsvar
```

To add a Blue Gene ION, it must be up and running. So a block that contains it needs to be booted. In addition, the SSH setup must be completed as described above. We test the procedure by adding a single ION, verifying its operation, and deleting it again. Afterwards we add all IONs to the GPFS cluster.

### Adding a single ION

After booting a block that contains the ION we want to add, we verify network connectivity through **ping**, verify that it can be accessed by **ssh**, and then add the node using **mmaddnode**:

```
# local command to boot a block for a user:
bglallocate R001_NF ibm010

ping -c 5 r00-m1-nf-4          # check that the node is
reachable                      #
ssh      r00-m1-nf-4 "rpm -qa|grep gpfs.base" # check that ssh to the node
works

mm1snode -a
#GPFS nodeset      Node list
#-----
# CLIENT           SN-b FEN-b

mmaddnode "r00-m1-nf-4:client-nonquorum"
#Mon Feb 20 10:51:00 CET 2006: mmaddnode: Processing node
r00-m1-nf-4.your-domain.org
#mmaddnode: Command successfully completed
#mmaddnode: Propagating the changes to all affected nodes.
#This is an asynchronous process.
```

```

mmlsnode -a
#GPFS nodeset      Node list
#-----
# CLIENT          SN-b FEN-b r00-m1-nf-4

```

```

mmstartup -w r00-m1-nf-4
#Mon Feb 20 12:27:32 CET 2006: mmstartup: Starting GPFS ...

```

**Note:** When we first ran `mmaddnode` to add an ION, we received the following error:

```

mmaddnode "r00-m1-nf-4:client-nonquorum"
#mmaddnode: Unexpected output from the 'host -t a r00-m1-nf-4' command:
#Host r00-m1-nf-4 not found: 3(NXDOMAIN)
#mmaddnode: Incorrect node name r00-m1-nf-4 specified for command.
#mmaddnode: mmaddnode quitting. None of the specified nodes are valid.

```

This is caused by the fact that `mmaddnode` uses the Linux `host` command to resolve the IP name of the node to be added, and the Linux `host` command did not check the `/etc/hosts` file but did a DNS request instead. After adding all IONs to DNS, `mmaddnode` worked without problems.

We can also log in to the ION and interactively check its state:

```

ssh r00-m1-nf-4
#BusyBox v1.00-rc2 (2006.01.09-19:48+0000) Built-in shell (ash)
#Enter 'help' for a list of built-in commands.

$ ps -ef|grep gpfs
#root      1017      1  0 11:27 ?          00:00:00 [gpfsSwapdKproc]
#root      1161    1159  0 11:28 tty1     00:00:00 grep gpfs
$ ps -ef|grep mmfs
#root      950       1  0 11:27 ?          00:00:00 /bin/ksh
/usr/lpp/mmfs/bin/runmmfs
#root      1072     950  0 11:27 ?          00:00:00 /usr/lpp/mmfs/bin//mmfsd
#root      1163    1159  0 11:28 tty1     00:00:00 grep mmfs

$ echo all GPFS commands are in /usr/lpp/mmfs/bin as usual...
$ cd /usr/lpp/mmfs/bin
$ ./mmlscluster
$ ./mmlsnode -a
$ ./mmauth show
$ ./mmremotecluster show

$ cat /etc/fstab
#none      /proc proc    defaults      0 0
#/dev/home1 /home1 gpfs
    rw,dev=OWNING.your-domain.org: bglhome, ldev=bglhome, noauto 0 0
...
$ mount /home1

$ cd /var/mmfs ; ls
# . .. etc gen mmbackup ssl tmp

$ ls -al ssl # check if the multicluster setup has been replicated to the new
node:
#total 9

```

```

#drwx----- 3 root root 1024 Feb 20 09:51 .
#drwxr-xr-x 7 root root 1024 Feb 20 09:51 ..
#-rw-r--r-- 1 root root 181 Feb 20 09:51 authorized_keys
#lrwxr-xr-x 1 root root 7 Feb 20 09:51 id_rsa -> id_rsa1
#-rw-r--r-- 1 root root 457 Feb 20 09:51 id_rsa.cert
#-rw-r--r-- 1 root root 182 Feb 20 09:51 id_rsa.pub
#-rw----- 1 root root 493 Feb 15 16:43 id_rsa1
#-rw-r--r-- 1 root root 181 Feb 20 09:51 known_cluster.OWNING.your-domain.org
#-rw-r--r-- 1 root root 190 Feb 20 09:51 openssl.conf
#drwx----- 2 root root 1024 Feb 20 09:51 stage

```

```

$ cd /var/adm/ras ; ls
#mmfs.log.2006.02.20.11.27.32.R00-M1-NF-4 mmfs.log.latest

```

```

$ /usr/lpp/mmfs/bin/mmsshutdown
#Mon Feb 20 11:44:39 UTC 2006: mmsshutdown: Starting force unmount of GPFS file
systems
#Mon Feb 20 11:44:44 UTC 2006: mmsshutdown: Shutting down GPFS daemons
#Shutting down!
# 'shutdown' command about to kill process 1072
#Unloading modules from /usr/lpp/mmfs/bin
#Unloading module mmfs
#Unloading module mmfslinux
#Unloading module tracedev
#Mon Feb 20 11:44:48 UTC 2006: mmsshutdown: Finished
#
$ exit

```

So everything looks OK on the ION.

## Deleting a single ION

Before adding all IONs, we delete the ION that was added in the previous section to check if this works. One important difference to other GPFS nodes is the fact that the diskless IONs store their local `/var/` data in a special `/bgl/gpfsvar/<ipaddress-of-ion>/var/` directory that is NFS-mounted from the service node. This directory needs to be carefully checked when nodes are added and deleted, to ensure that no stale configuration information is left there which could confuse the bring-up scripts on the ION. On the SN, we delete the ION we just added and then check and delete the corresponding `/bgl/gpfsvar/<ipaddress-of-ion>/` directory:

```

# local script that kills jobs on a block and frees it
bgldeallocate R001_NF

mmlsnode -a
#GPFS nodeset      Node list
#-----
# CLIENT           SN-b FEN-b r00-m1-nf-4

mmdelnode r00-m1-nf-4
#Verifying GPFS is stopped on all affected nodes ...
#mmdelnode: Command successfully completed
#mmdelnode: Propagating the changes to all affected nodes.
#This is an asynchronous process.

mmlsnode -a
#GPFS nodeset      Node list

```

```
#-----
# CLIENT          SN-b FEN-b

grep -i r00-m1-nf-4 /etc/hosts
#222.444.666.96    R00-M1-NF-4.your-domain.org  R00-M1-NF-4

find /bgl/gpfsvar/222.444.666.96/ # print this ION's /var tree's contents
rm -rf /bgl/gpfsvar/222.444.666.96/ # delete this ION's /var tree
```

Now all state information for the ION is deleted and it can be safely re-added.

## Adding all IONs to the GPFS cluster

After having tested the procedure with a single ION, we can now add all the IONs to the GPFS cluster. To do this we create a file with the hostnames of all IONs, add these to DNS, boot a block that spans the whole Blue Gene system, and then add the IONs in a loop:

```
# local script that allocates a block to a user:
bglallocate ALL ibm010

cat /tmp/gpfs.BGL-ion | while read ION
do
    echo "adding $ION..."
    mmaddnode "$ION:client-nonquorum"
done

mmlsnode -a
```

It is also possible to pass a GPFS NodeFile to `mmaddnode`, using the `-n` option. This would result in the addition of all the IONs in one step. We prefer the node-by-node approach so we can more easily spot problems that may occur in the process. However, this has the disadvantage that it takes a longer time to complete, as the configuration data gets replicated to all nodes at each loop step.

To mount the GPFS file systems, we use a mount script in `$BGL_SITEDSTISTDIR` named `/bgl/dist/etc/rc.d/rc3.d/S41gpfs_mount`.

## Customizing the GPFS startup process for the IONs

Caused by the fact that the IONs are diskless, their startup process differs from normal Linux installations. Starting up SSH and GPFS on the IONs is triggered by files in `/etc/sysconfig/` that the administrator needs to create. Our rc-file creates the GPFS file as follows:

```
cat /bgl/dist/etc/rc.d/rc3.d/S10fzj_sysconfig
#!/bin/sh
. /proc/personality.sh
. /etc/rc.status

...

# Trigger startup of GPFS:
echo "# if GPFS_STARTUP=1 in this file," > /etc/sysconfig/gpfs
echo "# the driver's S40gpfs will start GPFS." >> /etc/sysconfig/gpfs
echo "# (SSHD will also be started via S16sshd)" >> /etc/sysconfig/gpfs
echo "GPFS_STARTUP=1" >> /etc/sysconfig/gpfs
echo "#GPFS_VAR_DIR=/bgl/gpfsvar" >> /etc/sysconfig/gpfs
echo "#GPFS_CONFIG_SERVER=\"\$BGL_SNIP\"" >> /etc/sysconfig/gpfs
```



This creates the `/etc/sysconfig/gpfs` file, which gets sourced by the GPFS startup script, and since `GPFS_STARTUP` is set to true, that script will start GPFS.

## Making the startup process more scalable

The GPFS startup script `S40gpfs` shipped with the Blue Gene system software includes a file-based check to verify that GPFS started correctly: an `mmfsup.scr` script is created, which will be called by GPFS during startup. This script touches a file, and after `mmauto1oad` the GPFS startup script checks whether this file exists or not:

```
# Create mmfsup script that will run when GPFS is ready
cat <<-EOF > /var/mmfs/etc/mmfsup.scr
#!/bin/sh
touch $upfile
EOF
chmod +x /var/mmfs/etc/mmfsup.scr

# Start GPFS and wait for it to come up
rm -f $upfile
/usr/lpp/mmfs/bin/mmauto1oad
```

We had some scalability problems with this approach: small partitions came up, but for larger partitions the rc-script occasionally reported a failure on a few of the IONs. However, GPFS was running OK on the “failed” nodes. In some instances we observed NFS problems, which in our environment affected two places:

- ▶ The `$upfile` that is touched is created in `/tmp/`. We had replaced `/tmp/` with an NFS-mounted, per ION directory and saw “NFS stale filehandle” errors on this directory.
- ▶ The `mmfsup.scr` script is created in the ION’s `/var/mmfs/` directory. This is not a local directory, but is NFS-mounted from the SN’s `/bgl/gpfsvar/bgl-ip-address/` directory.

Reverting to a local `/tmp/` directory reduced the frequency of the problem but didn’t completely eliminate it. Eventually we replaced the “touch-file” approach by a call to `mmgetstate` to monitor whether GPFS came up or not:

```
#MH# # commented out this check (turned out unreliable), we use mmgetstate
#MH# #
#MH# retries=300
#MH# until test -e $upfile
#MH# do sleep 2
#MH# let retries=$retries-1
#MH# if [ $retries -eq 0 ]
#MH# then ras_advisory "$0: GPFS did not come up on I/O node $HOSTID"
#MH# exit 1
#MH# fi
#MH# done
#MH# rm -f $upfile
#MH#
#MH# # BEGIN of mmgetstate check
ACTIVE=0
retries=10
until test $ACTIVE -eq 1
do sleep 2
/usr/lpp/mmfs/bin/mmgetstate \
|tail -1|/usr/bin/grep -vE "active|arbitrating"
ACTIVE=$?
let retries=$retries-1
```

```

        if [ $retries -eq 0 ]
        then ras_advisory "$0: GPFS did not come up on I/O node $HOSTID"
            exit 1
        fi
    done
    /usr/lpp/mmfs/bin/mmgetstate
#MH# # END of mmgetstate check

```

**Note:** You might also want to comment out the above creation of the `mmfsup.scr` callback to avoid the unnecessary NFS load caused by the creation of the callback script.

Using `mmgetstate` has proven to be reliable even when bringing up hundreds of IONs at once. The “arbitrating” state is typically observed when quorum is lost, for example when our single quorum node is “down”:

```

$ ./mmgetstate -a
Node number  Node name      GPFS state
-----
      1      SN-b          down
      2      FEN-b          down
     25      R00-M1-N4-4    arbitrating
     26      R00-M1-N5-4    arbitrating

```

**Tip:** To be able to patch the system-wide startup script, we copied it from `$BGL_DISTDIR` to `$BGL_SITEDISTDIR`, modified it, linked it into the startup sequence as `S39gpfs` *before* the system file starts up (`S40gpfs`), and near the end of our modified script we removed the `/etc/sysconfig/gpfs` file that triggers GPFS startup, to prevent the unmodified `S40gpfs` start-script from doing anything when it starts up later in the process.

## Tools for performance testing

After the GPFS setup is complete and the file system access is functional, it is a good idea to verify the performance of the solution. We split this into three steps:

1. Raw TCP/IP performance of the Ethernet connections.
2. GPFS performance for I/O operations originating on the IONs (testing only; end users do not have access to the IONs).
3. GPFS performance for end-user I/O, which originates on the compute nodes (CNs).

Obviously, the achievable bandwidth depends on the available networking and disk storage hardware. The main goal here is to verify that our configuration works as expected, and to identify potential bottlenecks relative to the baseline set by the available hardware.

**Note:** All performance measurements presented here were done during production operation, not in a dedicated benchmarking environment. So performance numbers are affected by other loads on the two clusters and the quoted performance numbers are for illustration only.

## TCP/IP performance tests

There are many tools to test TCP/IP performance. Two that we recommend are `nuttcp` and `Iperf` (see references). The following tests have been performed with the stable distribution of `nuttcp`, v5.1.3:

## Building nuttcp

Download the tool and build it, for pSeries-Linux on the SN and for AIX 5L on a node of the owning GPFS cluster. On the SN, put the executable into a directory under /bgl/ that is also accessible by the IONs and FEN(s). We used a /bgl/local/sbin/ directory for this purpose:

```
# build nuttcp on the SN...

mkdir ~/nuttcp
cd ~/nuttcp
for FILE in Makefile README examples.txt nuttcp.c nuttcp.cat nuttcp.html
do
  wget ftp://ftp.lcp.nrl.navy.mil/pub/nuttcp/stable/$FILE
done
make # gcc -O3 -I. -Imissing -o nuttcp-v5.1.3 nuttcp-v5.1.3.c
cp nuttcp-v5.1.3 /bgl/local/sbin/nuttcp
```

Repeat this on the AIX 5L side, and put the executable, for example, into /usr/local/sbin/nuttcp.

## Starting the nuttcp servers

There are two ways to start the server side of **nuttcp**: explicit invocation with **nuttcp -S**, or through inetd/xinetd. A sample xinetd config file can be found at the above site under ../stable/xinetd.d/nuttcp, but here we just use the explicit startup.

To test multiple streams, we start as many instances of **nuttcp -S** as we have Ethernet links on a node, using different port numbers. The “-P” option specifies the port used for data transfer, the “-p” option sets the port for control traffic.

```
nobody@nsd01: /usr/local/sbin/nuttcp -S -P5000 -p5001
nobody@nsd01: /usr/local/sbin/nuttcp -S -P5002 -p5003
nobody@nsd01: /usr/local/sbin/nuttcp -S -P5004 -p5005
nobody@nsd01: /usr/local/sbin/nuttcp -S -P5006 -p5007
```

On the IONs, we use an rc-file to start **nuttcp** on each ION in the blocks we want to use for testing. Note the case statement that evaluates the block name—this is useful especially for testing purposes on a system that is otherwise running a production workload:

```
cat /bgl/dist/etc/rc.d/rc3.d/S15nuttcp
#!/bin/sh
. /proc/personality.sh
. /etc/rc.status
case "$BGL_BLOCKID" in
# *)
# Disabled)
R000_NCDEF*)
    echo "Starting nuttcp -S"
    /bgl/local/sbin/nuttcp -S -P5000 -p5001
;;
esac
```

After booting, for example, the block R000\_NCDEF\_FAT, a **nuttcp** server is running on each of its IONs. Blocks that do not start with R000\_NCDEF are unaffected.

**Note:** Do not forget to stop the **nuttcp** servers when they are no longer needed, by killing the **nuttcp -S** process and/or disabling their startup on the IONs as shown in the above script.

## Performance measurements with nuttcp

On the client side, we normally run `nuttcp -w1m destination-ip`, which transfers data in blocks of 1 MB (our GPFS file system size) to the `nuttcp` server that runs on the server denoted by `destination-ip`. To measure multiple streams, we run multiple instances of the command in parallel. By default, the client transmits the data (“-t” option). It can also be called in receive mode, in which case it asks the server side to send data (“-r” option). A useful option for documenting multiple runs is “-l label” (capital i), which prints the label text at the beginning of the output line that shows the performance data. Our test script to run eight *transmit* streams in parallel is shown here:

```
root@nsd01::/tmp> cat nuttcp-t-8stream
H=~hostname~
./nuttcp -t -w1m -P5000 -p5001 -I$H:t:$1 $1 &
./nuttcp -t -w1m -P5000 -p5003 -I$H:t:$2 $2 &
./nuttcp -t -w1m -P5000 -p5005 -I$H:t:$3 $3 &
./nuttcp -t -w1m -P5000 -p5007 -I$H:t:$4 $4 &
./nuttcp -t -w1m -P5000 -p5009 -I$H:t:$5 $5 &
./nuttcp -t -w1m -P5000 -p5011 -I$H:t:$6 $6 &
./nuttcp -t -w1m -P5000 -p5013 -I$H:t:$7 $7 &
./nuttcp -t -w1m -P5000 -p5015 -I$H:t:$8 $8
```

A similar script to *receive* data from the server uses the “-r” option and “:r:” in the label text.

**Note:** By default, `nuttcp` transmits data for 10 seconds. The timer it uses does not seem to work correctly on the IONs. You can press Ctrl-C when you run `nuttcp` on an ION to stop it transferring data. Alternatively, work from the other end of the connection and use the “-r” option to change the direction of the I/O.

With sufficiently many `nuttcp` servers running, we can then verify the TCP/IP performance of each point-to-point connection, as well as aggregate TCP/IP performance on a server. Typical tests include:

- ▶ Testing the *destination-ip* localhost or 127.0.0.1

This probes the efficiency of the TCP/IP stack on the local machine. Depending on the server model, we typically observe between 6000 Mbps and 10000 Mbps, with 100%TX and over 50%RX busy times. On the IONs, lower performance is an indication of the 700 MHz CPU speed.

```
FEN:/bg1/local/sbin> ./nuttcp -w1m 127.0.0.1 # run it several times...
11917.0625 MB / 10.00 sec = 9994.6927 Mbps 99 %TX 73 %RX
```

```
root@nsd01::/usr/local/sbin> ./nuttcp -w1m localhost # run it several
times...
10934.9375 MB / 10.00 sec = 9172.7372 Mbps 100 %TX 73 %RX
```

```
root@ION $ ./nuttcp -w1m 127.0.0.1 # a ppc440 BG/L node (use ctrl-C to stop)
2458.4375 MB / 9.91 sec = 2081.7283 Mbps 53 %TX 45 %RX
```

- ▶ Single-stream performance between an NSD server and the FEN or SN

Since we do not use round-robin mode for the EtherChannels, a single stream will not utilize more than a single 1 Gbps Ethernet link even though there are multiple links in the channels. We can verify whether this one link is utilized effectively by running `nuttcp` between an NSD server and the FEN, using several runs of both transmit and receive mode:

```
nsd01:t:FEN-b: 1120.3252 MB / 10.01 sec = 938.9669 Mbps 14 %TX 35 %RX
```

```

nsd01:t:FEN-b: 1121.6722 MB / 10.01 sec = 940.0999 Mbps 14 %TX 35 %RX
nsd01:t:FEN-b: 1120.7954 MB / 10.01 sec = 939.3755 Mbps 13 %TX 11 %RX

nsd01:r:FEN-b: 1121.6939 MB / 10.01 sec = 939.5753 Mbps 18 %TX 37 %RX
nsd01:r:FEN-b: 1120.9504 MB / 10.01 sec = 939.1449 Mbps 18 %TX 37 %RX
nsd01:r:FEN-b: 1120.8271 MB / 10.01 sec = 938.9185 Mbps 18 %TX 37 %RX

```

The achieved bandwidth is around 940 Mbps, which is close to “wire speed”: With the default 1500-byte MTU size, the maximum TCP throughput is 941.482 Mbps; *jumbo frames* with 9000-byte MTU would increase this to roughly 990 Mbps.

► Single-stream performance between an NSD server and an ION

The performance of the single 1 Gbps link of an ION should also approach wire speed. However, we observe that the achieved maximum rate is lower than for the FEN, and that the CPU load on the ION is significantly higher. This is an indication that the TCP/IP bandwidth of the ION is CPU-limited. In addition, *transmit* rates are roughly 6% higher than *receive* rates (viewed from the NSD server):

```

nsd01:t:r00-m0-nf-4: 1008.3381 MB / 10.00 sec = 845.6231 Mbps 6 %TX 99 %RX
nsd01:t:r00-m0-nf-4: 1008.5152 MB / 10.00 sec = 845.7808 Mbps 6 %TX 100 %RX
nsd01:t:r00-m0-nf-4: 1008.1824 MB / 10.00 sec = 845.5076 Mbps 6 %TX 100 %RX

nsd01:r:r00-m0-nf-4: 948.5580 MB / 10.00 sec = 795.4084 Mbps 61 %TX 34 %RX
nsd01:r:r00-m0-nf-4: 947.9534 MB / 10.00 sec = 794.9104 Mbps 61 %TX 33 %RX
nsd01:r:r00-m0-nf-4: 950.2382 MB / 10.00 sec = 796.8169 Mbps 62 %TX 34 %RX

```

Using *jumbo frames* would considerably improve the CPU load, so if this is possible in your environment, we do recommend to use jumbo frames on the Blue Gene I/O network. Note that you also need to configure this in the DB2 tables that describe the Blue Gene system properties.

► Multiple streams between an NSD server and several IONs

This is the case that most closely resembles file system I/O, where several IONs of a parallel job do I/O to each of the NSD servers. To perform these tests we use the above `nuttcp-t-8stream` (and `nuttcp-r-8stream`) script to run eight streams in parallel over the 4-port EtherChannel. A typical output is shown here:

```

root@nsd01::/tmp> ./nuttcp-t-8stream \
  r00-m0-nc-3 r00-m0-nc-4 r00-m0-nd-3 r00-m0-nd-4 \
  r00-m0-ne-3 r00-m0-ne-4 r00-m0-nf-3 r00-m0-nf-4 | sort
nsd01:t:r00-m0-nc-3: 276.8828 MB / 10.03 sec = 231.4698 Mbps 2 %TX 25 %RX
nsd01:t:r00-m0-nc-4: 550.6213 MB / 10.02 sec = 461.1474 Mbps 4 %TX 51 %RX
nsd01:t:r00-m0-nd-3: 282.5860 MB / 10.03 sec = 236.3620 Mbps 2 %TX 25 %RX
nsd01:t:r00-m0-nd-4: 1010.4202 MB / 10.01 sec = 846.7955 Mbps 7 %TX 99 %RX
nsd01:t:r00-m0-ne-3: 1007.7137 MB / 10.00 sec = 845.3867 Mbps 7 %TX 99 %RX
nsd01:t:r00-m0-ne-4: 282.2201 MB / 10.02 sec = 236.2378 Mbps 2 %TX 25 %RX

```

```
nsd01:t:r00-m0-nf-3: 572.4909 MB / 10.01 sec = 479.6471 Mbps 5 %TX 52
%RX
nsd01:t:r00-m0-nf-4: 283.9076 MB / 10.03 sec = 237.4760 Mbps 2 %TX 26
%RX
```

This output clearly shows that in this instance two of the eight connections are running at the IONs' maximum speed over two of the ports, whereas the six other connections share the remaining two ports in the channel.

**Note:** The exact patterns vary from run to run because the AIX 5L EtherChannel uses `hashmode=src_dst_port`, and the source port of the connection is picked at random. This is also true for a GPFS connection where one side always uses port 1191 and the other side uses a random port.

When all the TCP/IP tests have been done and the results are consistent with the expected performance of the given hardware, the next step is to verify the file system performance.

## GPFS performance tests on the IONs

Before looking at application-level I/O performance, we perform measurements directly on the IONs. This shows the performance of the GPFS client on the ION without being affected by the hardware and software components that connect an ION with its Compute Nodes (CNs). The IONs are not accessible by end users, so this is only a testing scenario.

The bandwidth of the underlying GPFS file systems is much higher than the bandwidth of the 1 Gbps Ethernet link of a single ION, so the file system is not a bottleneck for this test.

### Building `iozone` for pSeries Linux and Blue Gene

We use the `iozone` benchmark for our file system performance tests on the IONs. It can be built from source without any modifications:

```
cd /tmp
wget http://www.iozone.org/src/current/iozone3_263.tar
tar zvf iozone3_263.tar
cd iozone3_263/src/current/
make linux-powerpc
cp -p iozone /bgl/local/sbin/iozone
```

Note that we did the build with a `make` target of `linux-powerpc`, not `linux-powerpc64`. This means that a 32-bit executable has been created, which runs on the 64-bit SN and FEN(s) as well as on the 32-bit IONs.

### Performance measurements with `iozone`

The `iozone` benchmark was originally developed for commercial workloads such as database I/O, and has many options. For HPC we are mainly interested in sequential, large block I/O. This can be achieved with `iozone` by using an invocation like the following:

```
cd /your-filesystem ; iozone -r1m -s5g -i 0 -i 1 -w
```

This does I/O in chunks of 1 MB to a file of 5 GB. The “-i 0” option invokes a sequential write/rewrite test, and the “-i 1” option performs sequential read/reread. By using the “-w” option the test file will not be erased after a test, so in our case the read test can use the test file that is left over from the write test. Running this multiple times on a single ION produces output like the following (only the line with performance data is shown):

```
5242880 1024 81844 84563 69549 69407
```

```
5242880    1024    82833    84937    68970    69719
...
```

This gives the filesize, blocksize, write/rewrite and read/reread rates. Again, writes from an ION to GPFS are performing better than reads (82 MB/s versus 70 MB/s), whereas read and write on the FEN perform roughly at the same rate.

Aggregate performance can be measured by running **iozone** in parallel on multiple IONs. Ideally this should scale linearly until the bandwidth of the network connections or storage subsystem is saturated.

## GPFS performance tests on the compute nodes

The I/O rates that can be achieved at the application level are the ultimate test of the parallel file system. In MPI-parallel applications, there are two general patterns of I/O:

- ▶ All MPI tasks send their data to MPI task 0 using MPI calls, and this one task performs the disk I/O. This obviously is not scalable with respect to bandwidth. On Blue Gene with its relatively weak individual nodes, this pattern should be changed to perform disk I/O in parallel from many (if not all) MPI tasks.
- ▶ Each MPI task does disk I/O. This approach is scalable with respect to bandwidth, and comes in two flavors:
  - Each task does I/O to its own individual file.
  - Each task does I/O to a section of one shared file.

While the former approach is generally working in many environments (for example, it will even work without a global file system if the files are only scratch data), it may cause problems when thousands (or tens of thousands) of files are created. In contrast, the shared-file approach needs file system support for locking sections of a shared file (GPFS provides this feature). It avoids the potentially problematic creation of huge numbers of files, but on the other hand it may incur some extra cost for file system operations like a file close(), which need to synchronize state across all tasks that have the shared file open.

To test the parallel I/O bandwidth as well as the performance of operations such as open() and close(), we use the IOR benchmark developed by Lawrence Livermore National Laboratory (LLNL).

### Building the LLNL parallel I/O benchmark IOR

The IOR benchmark is already adapted to Blue Gene. To build it, we only modified the Makefile to make sure it used the Blue Gene target:

```
cd /bglhome-local/admin/ibm010/
scp ...../IOR-2.8.10.tar.gz .
tar xzvf IOR-2.8.10.tar.gz
cd IOR-2.8.10/src/C/
vi Makefile
echo "edited Makefile to set uname to BGL, makefile.config to use mpicc"
```

To get a summary of IOR's options, invoke it with the "-h" option after it has been built.

### Performance measurements with IOR

Two different test cases were measured:

- ▶ The dependence of I/O rates through a *single* ION on the number of compute nodes that simultaneously perform I/O through this ION.

- Scaling of aggregate I/O bandwidth with the number of IONs in a block.

We used the following IOR invocation to set parameters like the blocksize, transfer size, file size, the number of MPI tasks participating in the I/O, and to control whether I/O is performed to a single shared file or to individual files for each process (“-F” option):

```
mpirun -partition $BLOCK -verbose 1 -cwd $DIR $DIR/IOR \
-i $REPETITIONS -b 1m -t 1m -s 512 [-F] -N $N
```

### Single ION performance

In the first test, the \$BLOCK is a partition with a single ION and 32 CNs in it, which has been preallocated. We use coprocessor mode and vary the number of compute nodes \$N. Each CN is writing a 512 MB chunk of data. For the small numbers of tasks tested here, there was no significant difference between the shared-file and individual-file modes. Only for larger task counts do the global operations like open() and close() become a significant factor.

Due to the way the ION and CNs communicate over the Blue Gene collective network, a single CN will typically not saturate an ION for reads. Figure 5 shows the read and write performance of a single ION when performing I/O from a varying number of compute nodes.

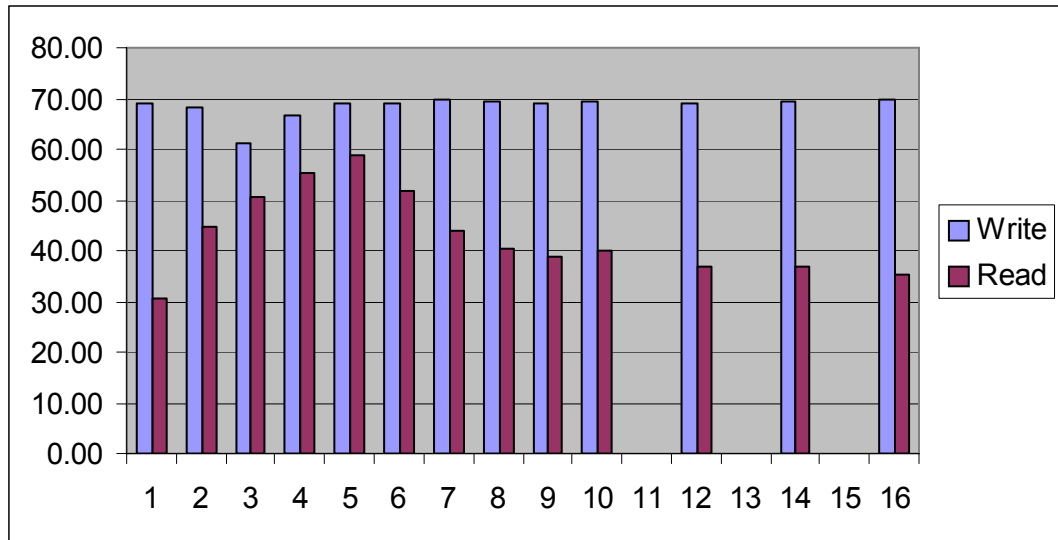


Figure 5 I/O performance of a single ION (MB/s versus number of CNs)

Write performance is around 70 MB/s. The read performance of a single CN is significantly less than the write performance. When multiple CNs are reading data in parallel, the aggregate read performance of the ION increases, but eventually gets lower again as larger numbers of CNs are added (data for 32 CNs is similar to that for 16 CNs).

**Note:** Small variations are caused by the fact that these runs are performed on a live system running a production workload. The performance data given here is for illustration of general trends only and should not be considered accurate with respect to absolute numbers.

### Multiple IONs performance

When adding more IONs, the aggregate GPFS performance should scale up linearly until the bandwidth of the network connections or storage subsystem is saturated. To test this, we allocate a midplane with 16 IONs, and then use a mapfile to run IOR on one CN per ION. In



our case the first CN in the 16 pSets of the 512-node midplane block had the following torus coordinates (using coprocessor mode, so the 4<sup>th</sup> dimension is always zero):

```
cat ./MAPFILE
0 0 0 0
4 0 0 0
0 4 0 0
4 4 0 0
0 0 2 0
4 0 2 0
0 4 2 0
4 4 2 0
0 0 4 0
4 0 4 0
0 4 4 0
4 4 4 0
0 0 6 0
4 0 6 0
0 4 6 0
4 4 6 0
```

**Note:** To find the location of the IONs on the torus network (more precisely the position of the first CN in an ION's *pSet*, since the IONs themselves are not on the torus network), one can use the Blue Gene personality data structure as described in Appendix B.2 of the *Unfolding Blue Gene* redbook.

Using this mapfile and a preallocated midplane block R001, the performance measurements can then be done as follows:

```
for N in 1 2 3 4 5 6 7 8 12 16
do
  mpirun -partition R001 -verbose 1 -mapfile ./MAPFILE -np $N \
    -cwd $DIR $DIR/IOR -N $N -i $COUNT -b 1m -t 1m -s 512 | tee
  results512.$N.txt
done
```

Figure 6 shows the resulting aggregate I/O bandwidth when increasing the number of tasks, with task placement through a mapfile so each task uses a different ION.

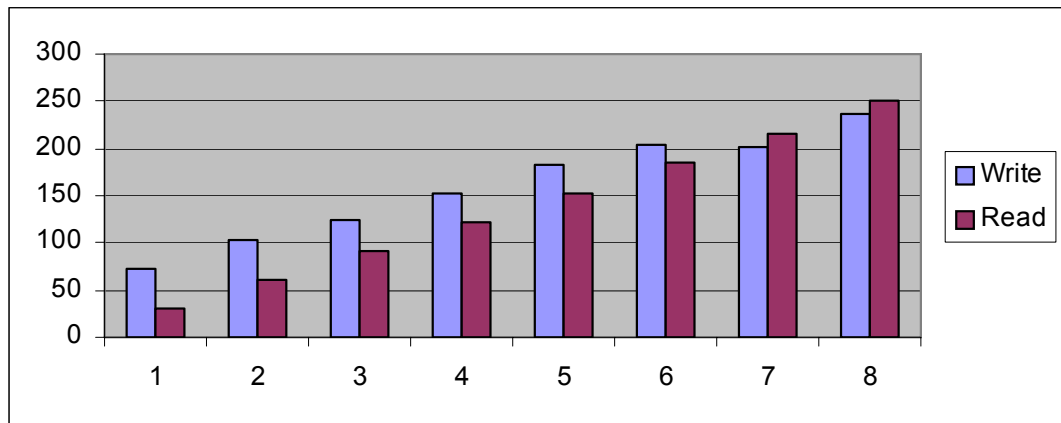


Figure 6 I/O performance scaling with the number of IONs in a block (MB/s versus number of IONs)

As for the single-ION measurements, read bandwidth for a single ION is smaller than write bandwidth, since we are only using one CN per ION. Running, for example, 4 MPI tasks per pSet should improve that number.

## ION memory considerations

It is useful to experiment with the various buffer sizes on the ION, but one needs to take care to not overrun the ION's main memory of 512 MB. The two main consumers of memory on the ION are the GPFS daemon (especially its pagepool), and the CIOD daemon, which handles I/O to and from the compute nodes. Here is a footprint from an ION in a block with a pSet size of 32 (one ION handles 32 CNs):

```
$ ps aux|grep -E "USER|mmfs|ciod"
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         32  0.0  0.0      0     0 ?        S      13:57   0:00 [rpciod]
root        624  0.0  0.3   3040  2040 ?        S<     13:57   0:00 /bin/ksh
/usr/lpp/mmfs/bin/runmmfs
root        819  0.2 27.5 539804 142352 ?        S<L    13:57   0:02
/usr/lpp/mmfs/bin/mmfsd
root       1286  0.0 13.0  78052 67576 ?        SL     13:58   0:00
/sbin.rd/ciod.440
root       1403  0.0  0.1   1608   520 tty0     S+     14:09   0:00 grep -E
USER|mmfs|ciod
```

```
$ /usr/lpp/mmfs/bin/mmlsconfig |grep pagepool
pagepool 128M
```

```
$ cat /etc/sysconfig/ciod
export CIOD_RDWR_BUFFER_SIZE=1048576
export DEBUG_SOCKET_STARTUP=ALL
```

The memory footprint from a block with pSet size 16, where the CIOD on one ION correspondingly only handles half the number of CNs, shows a proportionally smaller memory consumption of CIOD:

```
$ ps aux|grep -E "USER|mmfs|ciod"
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         32  0.1  0.0      0     0 ?        S      14:13   0:00 [rpciod]
root        624  0.3  0.3   3040  2040 ?        S<     14:13   0:00 /bin/ksh
/usr/lpp/mmfs/bin/runmmfs
root        819  2.4 27.4 539804 142124 ?        S<L    14:14   0:01
/usr/lpp/mmfs/bin/mmfsd
root       1286  0.2  6.7  45156 34680 ?        SL     14:14   0:00
/sbin.rd/ciod.440
root       1302  0.0  0.1   1608   520 tty0     S+     14:14   0:00 grep -E
USER|mmfs|ciod
```

Obviously, the ION allocates *two* times the CIOD\_RDWR\_BUFFER\_SIZE for each of the CNs that it serves (one buffer for each of the two processors on the CN), so it has sufficient buffer space if virtual node mode is used.

## Summary

In this Redpaper we show the necessary steps to add an IBM System Blue Gene supercomputer to a heterogeneous GPFS multicluster environment. Particular attention is

paid to the networking setup and the peculiarities in configuring the various components for the Blue Gene I/O nodes. We also demonstrate how to do basic performance measurements to verify the setup, and the resulting data shows reasonable scaling of the I/O bandwidth with both the number of IONs and the number of MPI tasks per ION.

Three areas could be investigated further to improve performance and/or manageability:

- ▶ *Jumbo frames* could be used on the Blue Gene I/O network. This would improve TCP/IP performance of the IONs by reducing their CPU load.
- ▶ The new Blue Gene system software *v1r3* contains improvements of various components, including modifications of the *ciod* daemon on the IONs, which should improve the read performance of a single CN per ION.
- ▶ *GPFS Version 3.1* includes support for subnets, which drastically simplifies the routing setup for GPFS multicluster environments, as well as other performance improvements.

These areas have not been covered here, but will be investigated as soon as the customer environment that has been used for the study has been migrated to the new Blue Gene and GPFS software releases.

## Acknowledgements

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

Dino Quintero  
IBM International Technical Support Organization, Poughkeepsie Center

Jutta Docter, Klaus Wolkersdorfer, Olaf Mextorf, Michael Stephan, Lothar Wollschläger,  
Franz-Josef Schönebeck, Monika Ollech  
Forschungszentrum Jülich (FZJ)

Gautam Shah, Kalyan Gunda  
IBM Poughkeepsie

Tom Engelsiepen  
IBM Almaden

Chris Stone  
IBM Hursley

Karsten Kutzer  
IBM Germany

## References

### *IBM Redbooks*

The following Blue Gene Redbooks™ are available at:

<http://www.redbooks.ibm.com/>

(search for “Gene”):

- ▶ *Blue Gene System Administration*, SG24-7178
- ▶ *Blue Gene Hardware Overview and Planning*, SG24-6796

- ▶ *Blue Gene Application Development*, SG24-7179
- ▶ *Blue Gene Problem Determination*, SG24-7211
- ▶ *Unfolding Blue Gene*, SG24-6686

## Other publications

*General Parallel File System HOWTO for the IBM System Blue Gene Solution*, SC23-5192-04. Shipped with the GPFS 2.3 for Blue Gene RPQ software.

*I/O Node Startup and Shutdown Scripts*. Shipped with the Blue Gene v1r2 system software. After installation it is available at:

`/bgl/BlueLight/ppcfloor/docs/ionode.README`

The GPFS 2.3 documentation is available from the Cluster Information Center at:

<http://publib.boulder.ibm.com/infocenter/clresctr/index.jsp>

and includes:

- ▶ *Concepts, Planning and Installation Guide*, GA22-7968-02
- ▶ *Administration and Programming Reference*, SA22-7967-02
- ▶ *Problem Determination Guide*, GA22-7969-02
- ▶ *Data Management API Guide*, GA22-7966-02

*Linux Ethernet Bonding Driver HOWTO* - Part of the Linux distribution, available at:

`/usr/src/linux-2.6.5-7.*/Documentation/networking/bonding.txt`

*Linux /proc/sys/net/ipv4/\* variables documentation* - Part of the Linux distribution, available at:

`/usr/src/linux-2.6.5-7.*/Documentation/networking/ip-sysctl.txt`

*AIX 5L System Management Guide: Communications and Networks*, section “EtherChannel and IEEE 802.3ad Link Aggregation” - Available from the AIX® 5L Information Center at:

[http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.doc/aixbman/commadm/tcp\\_etherchannel.htm](http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.doc/aixbman/commadm/tcp_etherchannel.htm)

Cisco Systems, Inc.: *Catalyst 6500 Series Switch Software Configuration Guide (Software Release 8.4)*, chapter 6 “Configuring EtherChannel”; Cisco Part Number OL-6210-01.

Cisco Systems, Inc.: *Catalyst 6500 Series Switch Command Reference (Release 8.4)*; Cisco Part Number OL-6244-01.

## Online resources

*IBM Journal of Research and Development*, Volume 49, Issue 2/3: “Blue Gene”. Available from:

<http://www.research.ibm.com/journal/rd49-23.html>

Jülich BlueGene/L (“JUBL”) Web pages at:

<http://www.fz-juelich.de/zam/ibm-bgl/>

DEISA.org project home page at:

<http://www.deisa.org/>

Links to some performance measurement tools that were used during the case study:

- ▶ The **nuttcp** tool at:

<ftp://ftp.lcp.nrl.navy.mil/pub/nuttcp/>

- ▶ The **Iperf** tool at:

<http://dast.nlanr.net/Projects/Iperf/>

- ▶ The **iozone** tool at:

<http://www.iozone.org/>

- ▶ The **ior** benchmark at:

<http://www.llnl.gov/asci/purple/benchmarks/limited/ior/>



# 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 that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right 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 U.S.A.*

**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.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

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 IBM's application programming interfaces.

This document created or updated on October 19, 2006.




Send us your comments in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:  
[ibm.com/redbooks](http://ibm.com/redbooks)
- ▶ Send your comments in an email to:  
[redbook@us.ibm.com](mailto:redbook@us.ibm.com)
- ▶ Mail your comments to:  
IBM Corporation, International Technical Support Organization  
Dept. HYTD Mail Station P099  
2455 South Road  
Poughkeepsie, NY 12601-5400 U.S.A.

## Trademarks

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

Redbooks (logo) ™  
eServer™  
pSeries®  
AIX®

Blue Gene®  
DB2®  
IBM®  
OpenPower™

POWER™  
POWER4+™  
Redbooks™  
System p™

The following terms are trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.