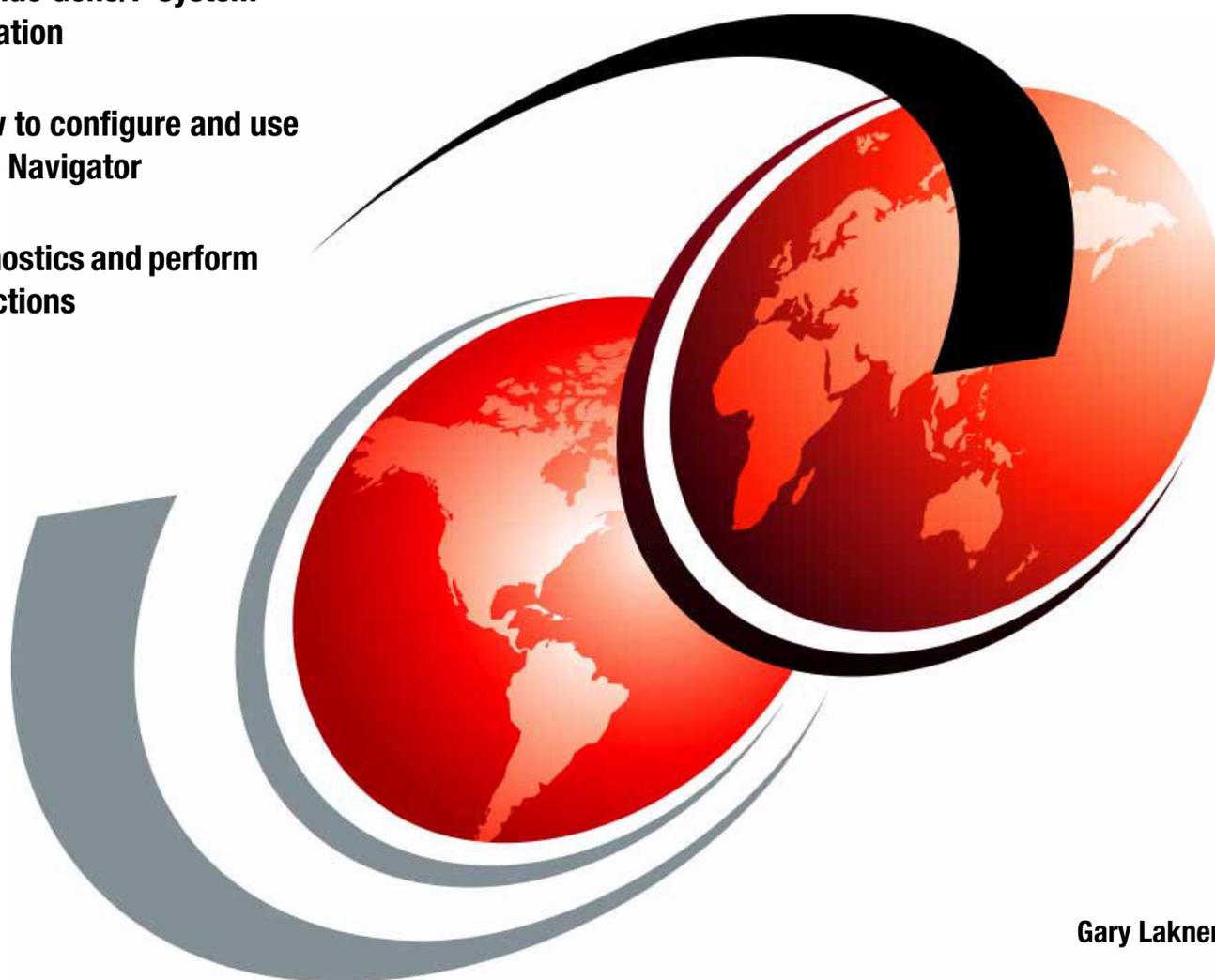


IBM System Blue Gene Solution: Blue Gene/P System Administration

Perform Blue Gene/P system
administration

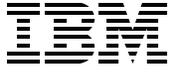
Learn how to configure and use
Blue Gene Navigator

Run Diagnostics and perform
Service Actions



Gary Lakner

Redbooks



International Technical Support Organization

**IBM System Blue Gene Solution: Blue Gene/P System
Administration**

June 2008

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

Second Edition (June 2008)

| This edition applies to Version1, Release 2, Modification 0 of Blue Gene/P(product number 5733-BGP).

This document created or updated on May 30, 2008.

Note: This book is based on a pre-GA version of a product and may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this redbook for more current information.

Contents

Notices	vii
Trademarks	viii
Preface	ix
The team that wrote this book	ix
Become a published author	x
Comments welcome	xi
Summary of changes	xiii
June 2008, Second Edition	xiii
.....	xiii
Chapter 1. The evolution of Blue Gene	1
1.1 Hardware changes	2
1.1.1 Packaging	2
1.1.2 Power requirements	3
1.1.3 Cooling requirements	4
1.1.4 Compute and I/O node	4
1.1.5 Networking updates	5
1.2 Software improvements	5
1.2.1 Location identification	5
1.2.2 Database	7
1.2.3 Reliability, Availability, Serviceability (RAS) infrastructure	8
1.2.4 Blocks	8
1.2.5 Threading	8
1.2.6 Job modes	9
1.2.7 Control System	10
1.2.8 Proxy replaced	10
1.2.9 CIODB	10
1.2.10 Midplane Management Control System	11
1.2.11 Hardware monitoring	11
1.2.12 Client/server mpirun	12
1.2.13 Bridge APIs	13
1.2.14 Navigator updates	13
1.2.15 Parts replacement	14
1.2.16 Diagnostics	15
Chapter 2. Blue Gene Navigator	17
2.1 Blue Gene Navigator overview	18
2.1.1 Setup	18
2.1.2 Navigator options	21
2.2 Common functions	24
2.2.1 User preferences and session data	34
2.3 Using the Navigator	35
2.3.1 Health Center	35
2.3.2 Jobs	38
2.3.3 Blocks	39
2.3.4 MMCS Console	39
2.3.5 RAS Event Log	41

2.3.6	Job History	43
2.3.7	System Logs	44
2.3.8	Utilization	46
2.3.9	Midplane Activity	47
2.3.10	Environmental Queries	48
2.3.11	Hardware Browser	49
2.3.12	Link Summary	52
2.3.13	Block Builder	53
2.3.14	Diagnostics	54
2.3.15	Service Actions	55
2.3.16	BG/P Master	55
2.3.17	RAS Message Types	56
2.4	Customizing Blue Gene Navigator	57
2.4.1	Adding resource links	57
2.4.2	Chart Plug-ins	58
2.4.3	Problem determination	61
Chapter 3. BGPMaster		63
3.1	BGPMaster	64
3.1.1	BGPMaster configuration file	64
3.1.2	BGPMaster command	64
3.1.3	BGPMaster's Navigator interface	67
3.2	mcServer	67
3.2.1	Starting mcServer	68
3.2.2	Stopping mcServer	69
3.2.3	Logging	69
3.3	Midplane Management Control System	70
3.3.1	Starting MMCS server	70
3.3.2	Stopping MMCS server	70
3.3.3	Logging	71
3.4	The mpirun daemon	71
3.4.1	Configuring mpirun	72
3.4.2	Environment variables	73
3.4.3	Front End Node	74
3.4.4	Starting the mpirun daemon	74
3.4.5	Stopping the mpirun daemon	74
3.4.6	Logging	74
3.5	Navigator server	74
3.5.1	Starting the navigator_server process	75
3.5.2	Stopping the navigator_server	75
3.5.3	Logging	75
3.6	Real-time	75
3.6.1	Configuring Real-time	76
3.6.2	Starting the realtime_server process	76
3.6.3	Stopping the Real-time server	77
Chapter 4. Blocks		79
4.1	Blocks	80
4.2	Blue Gene Navigator	81
4.3	Midplane Management Control System	85
Chapter 5. Midplane Management Control System		91
5.1	MMCS components	92
5.2	Starting the MMCS server	93

5.3 Using MMCS console	93
5.3.1 Starting an MMCS console session	93
5.3.2 Allocating blocks	94
5.3.3 Starting jobs	94
5.4 MMCS commands	95
5.5 Output	102
5.5.1 Command and response	102
5.5.2 Mailbox output	103
5.5.3 Job output	103
5.5.4 Redirection	103
5.6 CIOD	104
5.7 Environmental Monitor	104
Chapter 6. High Throughput Computing	111
6.1 How it works	112
6.2 Security	113
6.2.1 Submit client to submit mux	113
6.2.2 Submit mux to submit server authentication	113
6.2.3 Submit mux to HTC job controller	113
6.3 Setting up HTC	114
6.3.1 startsubmitserver (on the Service Node)	114
6.3.2 bgpmaster.init	114
6.3.3 Submit mux on the submit nodes	115
6.4 MMCS Console commands	116
Chapter 7. RAS Messages	117
7.1 Blue Gene/P RAS	118
7.2 RAS event message flow	118
7.3 Anatomy of a RAS message	119
7.4 Navigator interface to the RAS database	121
Chapter 8. Diagnostics	125
8.1 System Diagnostics	126
8.2 Diagnostic test cases	126
8.3 Running Diagnostics	130
8.3.1 Navigator's Diagnostics interface	131
8.3.2 Starting a Diagnostic run	134
8.3.3 Running Diagnostics from the command line	136
Chapter 9. Service Actions	139
9.1 Service Actions overview	140
9.1.1 Service Action logs	140
9.2 Preparing the hardware for service	141
9.2.1 Command line Service Actions	141
9.2.2 Starting Service Actions from the Navigator	142
9.3 Cycling power	146
Chapter 10. Blue Gene/P tools	149
10.1 Security Administration tool	150
10.2 Coreprocessor tool	151
10.2.1 Starting the Coreprocessor tool	151
10.2.2 Debugging live compute node problems	154
10.2.3 Saving your information	157
10.2.4 Debugging live I/O node problems	158

10.2.5 Debugging core files	158
Chapter 11. Configuring the I/O nodes	161
11.1 Using the I/O node startup and shutdown scripts	162
11.1.1 BusyBox	162
11.1.2 Site customization directory and the ramdisk	162
11.1.3 Startup flow	162
11.1.4 Shutdown flow.	163
11.1.5 Shell variables from personality info	163
11.1.6 I/O node log file.	164
11.2 Configuring I/O node services.	165
11.2.1 Configuring CIO daemon	165
11.2.2 Configuring Network Time Protocol	167
11.2.3 Configuring the Syslog daemon	167
11.3 I/O node performance tips.	168
11.3.1 Configuring the CIOD buffer size	168
11.3.2 Configuring Network File System	169
11.3.3 Performance tips for the NFS file server.	169
11.4 Typical site customization	170
11.4.1 Additional GPFS notes	173
Chapter 12. Power management	175
12.1 Blue Gene/P power management.	176
12.2 Invoking power management	176
12.2.1 Reactive power management	176
12.2.2 Proactive power management	176
12.3 RAS messages	177
Chapter 13. Cluster Systems Management for Blue Gene/P.	179
13.1 Overview	180
13.2 Monitoring the Blue Gene/P database with CSM	180
13.3 Customizing CSM Blue Gene monitoring capabilities	182
13.3.1 Defining your own CSM monitoring constructs	184
13.3.2 Miscellaneous related information.	186
Appendix A. Statement of completion	189
Appendix B. Blue Gene/P hardware naming convention	191
Hardware naming conventions used in Blue Gene/P.	192
Glossary	197
Related publications	201
IBM Redbooks publications	201
How to get IBM Redbooks	201
Help from IBM	201
Index	203

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 illustrate 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.

Trademarks

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

Redbooks (logo) ®
eServer™
AIX®
Blue Gene/L™
Blue Gene/P™

Blue Gene®
DB2®
General Parallel File System™
GPFS™
IBM®

LoadLeveler®
PowerPC®
Redbooks®
System i™

The following terms are trademarks of other companies:

Snapshot, and the Network Appliance logo are trademarks or registered trademarks of Network Appliance, Inc. in the U.S. and other countries.

Java, Power Management, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Internet Explorer, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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.

Preface

This book is one in a series of IBM® publications written specifically for the IBM System Blue Gene® supercomputer, Blue Gene/P™, which is the second generation of massively parallel supercomputer from IBM in the Blue Gene series. It provides an overview of the system administration environment for Blue Gene/P. It is intended to help administrators understand the tools that are available to maintain this system.

This book explains briefly the evolution of the system, from Blue Gene/L to the new Blue Gene/P. It details Blue Gene Navigator, which has grown to be a full featured Web-based system administration tool on Blue Gene/P. The book also describes many of the day-to-day administrative functions, such as running Diagnostics, performing Service Actions, and monitoring hardware. There are also sections to cover BGPMaster and the processes that it monitors. The final chapters of the book cover the tools that are shipped with the system and details about configuring communications with the I/O nodes and Power Management™ features.

The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Poughkeepsie Center.

Gary Lakner is a Staff Software Engineer for IBM Rochester on assignment in the ITSO. He is a member of the Blue Gene/L™ Support Team in the IBM Rochester Support Center, where he specializes in both Blue Gene hardware and software, as well as performs customer installations. Prior to joining the Blue Gene/L team, he supported TCP/IP communications on the IBM eServer™ System i™ server. Gary has been with IBM since 1998.

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

Tom Gray
Todd Kelsey
Gary Mullen-Schultz
Jenifer Servais
Craig Schmitz
Carlos Sosa
International Technical Support Organization, Rochester Center

Mike Blocksome
Lynn Boger
Kathy Cebell
Todd Inglett
Tom Gooding
Nicholas Goracke
Tom Liebsch
Chris Marroquin
Pat McCarthy
Sam Miller
Mark Megerian
Sam Miller
Mike Mundy

Roy Musselman
Tom Musta
Mike Nelson
John Orbeck
Jeff Parker
Ruth Poole
Joan Rabe
Joseph Ratterman
Don Reed
Harold Rodakowski
Richard Shok
Brian Smith
Will Stockdell
Jim Van Oosten
Bin Ye
IBM Rochester

Marty Fullam
IBM Poughkeepsie

Special thanks to Brant Knudson, Tom Budnik, and Paul Allen of IBM Rochester for their time and contributions to the project.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- ▶ Send your comments in an e-mail to:

redbooks@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

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-7417-01
for *IBM System Blue Gene Solution: Blue Gene/P System Administration*
as created or updated on May 30, 2008

June 2008, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information described in the following sections.

New information

- ▶ RAS Messages (Chapter 7., "RAS Messages" on page 117)
- ▶ High Throughput Computing Chapter 6., "High Throughput Computing" on page 111)
- ▶ V1R2M0 Diagnostic updates (Chapter 8., "Diagnostics" on page 125)

Modified information

- ▶ Blue Gene Navigator updates (Chapter 2., "Blue Gene Navigator" on page 17)
- ▶ Steps to cycle power has been changed (9.3, "Cycling power" on page 146)



1

The evolution of Blue Gene

This chapter highlights some of the significant changes made in the evolution process of Blue Gene/L to Blue Gene/P.

1.1 Hardware changes

Blue Gene/P will keep the familiar, slanted profile that was introduced with Blue Gene/L. However the increased compute power requires increased airflow, resulting in a larger footprint. Each of the air plenums on Blue Gene/P are just over 10 inches wider than the previous model's plenums. Additionally, each Blue Gene/P rack is approximately four inches wider. There are two additional Bulk Power Modules mounted in the Bulk Power enclosure on the top of the rack. Rather than a circuit breaker style switch there is an on/off toggle switch to power the rack on.

1.1.1 Packaging

Figure 1-1 illustrates the packaging of Blue Gene/L.

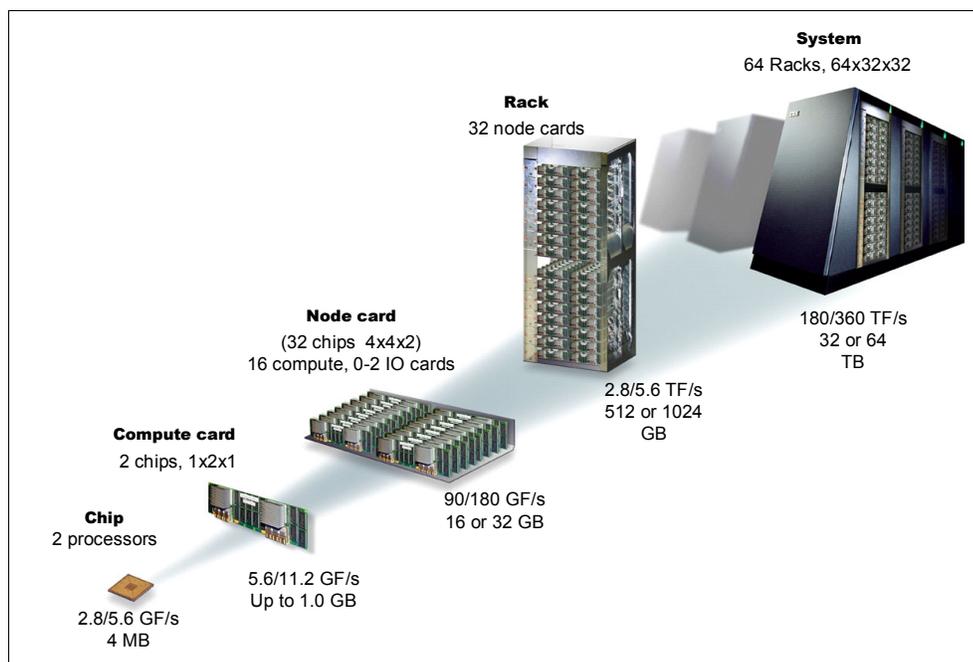


Figure 1-1 Blue Gene/L packaging

Figure 1-2 shows how Blue Gene/P has been packaged.

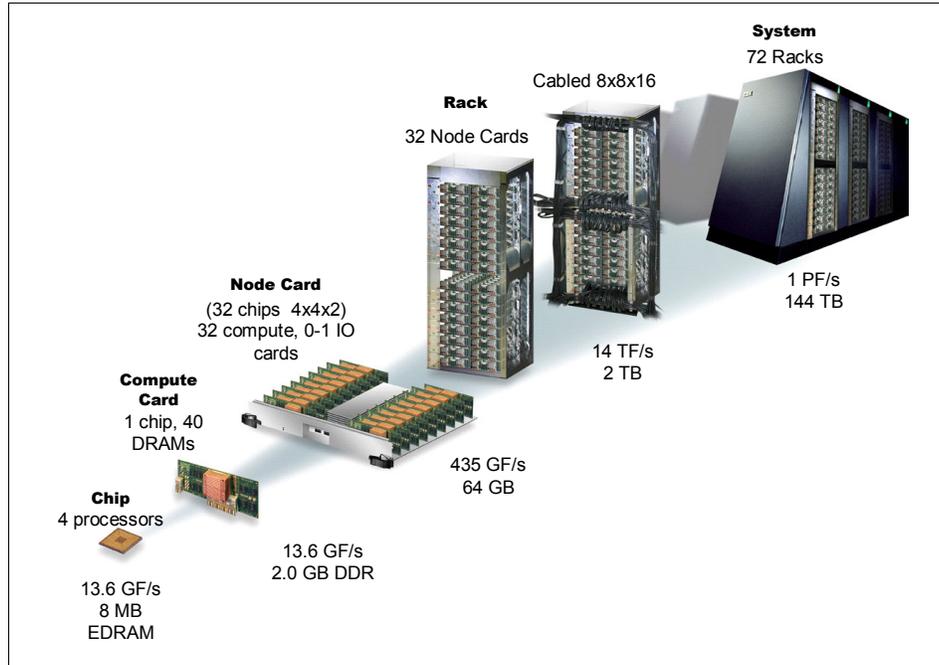


Figure 1-2 Blue Gene/P packaging

The changes start at the very bottom of the chain. Each chip is made of four processors rather than Blue Gene/L's two. As you step up to the next level, you notice that there is only one chip on each of the compute (processor) cards. This design is easy to maintain with less waste. Replacing a Blue Gene/L compute node because of a single failed processor meant discarding one usable chip (because they are packaged with two per card). The design of Blue Gene/P only has one chip per processor card, eliminating the disposal of a good chip when a Compute card is replaced.

There are still thirty-two chips on each node card, but now the maximum number of I/O nodes per node card will be two, so there are only two ethernet ports on the front of each node card. Like Blue Gene/L there are two midplanes per rack. The lower midplane is considered to be the master. Each midplane drives one Service card, four Link cards and sixteen Node cards. Additionally there are twenty fan modules that pull cooled air across the cards that plug into the rack's midplanes.

1.1.2 Power requirements

Each Blue Gene/P rack has its own independent supply, which can either be set for 480V 3 phase input and has a 100A line cord with a plug which is connected to a dedicated circuit breaker, or is wired for 200V/400V operation with a 175A line card which is hard wired to an appropriately sized circuit breaker. On the rack, there are no circuit breakers but there is an ON-OFF switch at the top of the exhaust plenum. 48V DC power is supplied to the rack using nine 5KW wide-ranging power supplies which cover both the US 208V and 480V 3phase AC, and the 200V power used elsewhere. The ninth power supply is redundant, eight are enough to supply power to the rack. From that point we use local, point of load DC-DC power supplies whenever the local power consumption is of order 50W or more.

1.1.3 Cooling requirements

Each Blue Gene/P rack is cooled using 60 (20 sets of 3) 120mm fans. These fans operate at a maximum speed of 6800 rpm pulling air across the midplane. Blue Gene/L has the same configuration of fans but the peak speed is 6000 rpm. As with later versions of Blue Gene/L the fans can be fine tuned to adjust the amount of airflow within the various areas inside the rack.

Also, as we mentioned earlier, the racks are wider to provide better airflow across the cards in the rack.

1.1.4 Compute and I/O node

The Compute nodes and I/O nodes in Blue Gene/L were two unique pieces of hardware. In Blue Gene/P the two parts, although performing different functions, are actually interchangeable. Table 1-1 shows a comparison of the compute nodes.

Table 1-1 Comparison of Blue Gene/L and Blue Gene/P nodes

Feature	Blue Gene/L	Blue Gene/P
Cores per node	2	4
Core Clock Speed	700 MHz	850 MHz
Cache Coherency	Software managed	SMP
Private L1 cache	32 KB per core	32 KB per core
Private L2 cache	14 stream prefetching	14 stream prefetching
Shared L3 cache	4 MB	8 MB
Physical Memory per Node	512 MB - 1 GB	2 GB
Main Memory Bandwidth	5.6 GBps	13.6 GBps
Peak Performance	5.6 GFlops per node	13.6 GFlops per node
Network Topologies	Blue Gene/L	Blue Gene/P
Torus Network		
Bandwidth	2.1 GBps	5.1 GBps
Hardware Latency (nearest neighbor)	200 ns (32B packet) and 1.6 μ s (256B packet)	100 ns (32B packet) and 800 ns (256B packet)
Global Collective Network		
Bandwidth	700 MBps	1.7 GBps
Hardware Latency (Round trip worst case)	5.0 μ s	3.0 μ s
Full System (72 rack comparison)		
Peak Performance	410 TFlops	~1 PFlops
Power	1.7 MW	~2.3 MW

The Compute Nodes contain four PowerPC® 450 processors with 2 GB of RAM and run a lightweight kernel to execute user-mode applications only. Typically, all four cores are used

for computation either in Dual Node Mode, Virtual Node Mode, or Symmetric Multiprocessor Mode. Data is moved between the compute and I/O nodes over the Global Collective network.

The I/O Nodes run an embedded Linux® kernel with minimal packages required to support an NFS client, Ethernet network connections and to act as a gateway for the Compute Nodes in their respective rack to the external world. The I/O nodes present a subset of standard Linux operating interfaces to the user. The 10 Gigabit Ethernet interface of the I/O Nodes is connected to the core Ethernet switch.

1.1.5 Networking updates

Blue Gene/P operates with the same five basic networks that were available on Blue Gene/L. The three-dimensional Torus, collective, and global barrier networks provide the high performance intra-application communications. The Functional network will allow the I/O nodes to send data to and receive data from the outside world while the Service (or Control) network provides a link from the Service node directly to the hardware.

To offload message handling overhead from the cores, injection and reception Direct Memory Access (DMA) has been added to Blue Gene/P. In addition to reducing the load on the core, it is expected that network blockages will be prevented if the memory queues are not drained fast enough.

Blue Gene/L provided a 1Gb Functional Ethernet while Blue Gene/P has been upgraded to a 10Gb Ethernet. The new Functional network is full duplexing and implements IPv4 checksums for both transmit and receive paths. On Blue Gene/P the Functional network can consist of up to 32 links per midplane and can transmit up to 300 meters over 50mm multimode fiber cable.

1.2 Software improvements

From a software perspective the overall design of Blue Gene/P is relatively unchanged from Blue Gene/L. Many of the changes implemented were made to accommodate the new hardware and enhance existing features.

1.2.1 Location identification

Location strings are used to identify hardware in the Blue Gene rack. Blue Gene/L used multiple methods to locate hardware depending on whether it was being referred to by the baremetal code or the software code. For example, in Blue Gene/L you might see R001, or R00-M1, both referring to the same midplane. With Blue Gene/P the references to the various

pieces of hardware are all consistent. Figure 1-3 shows the convention used for Blue Gene/P. These locations can also be used as regular expressions in MMCS commands such as `write_con`.

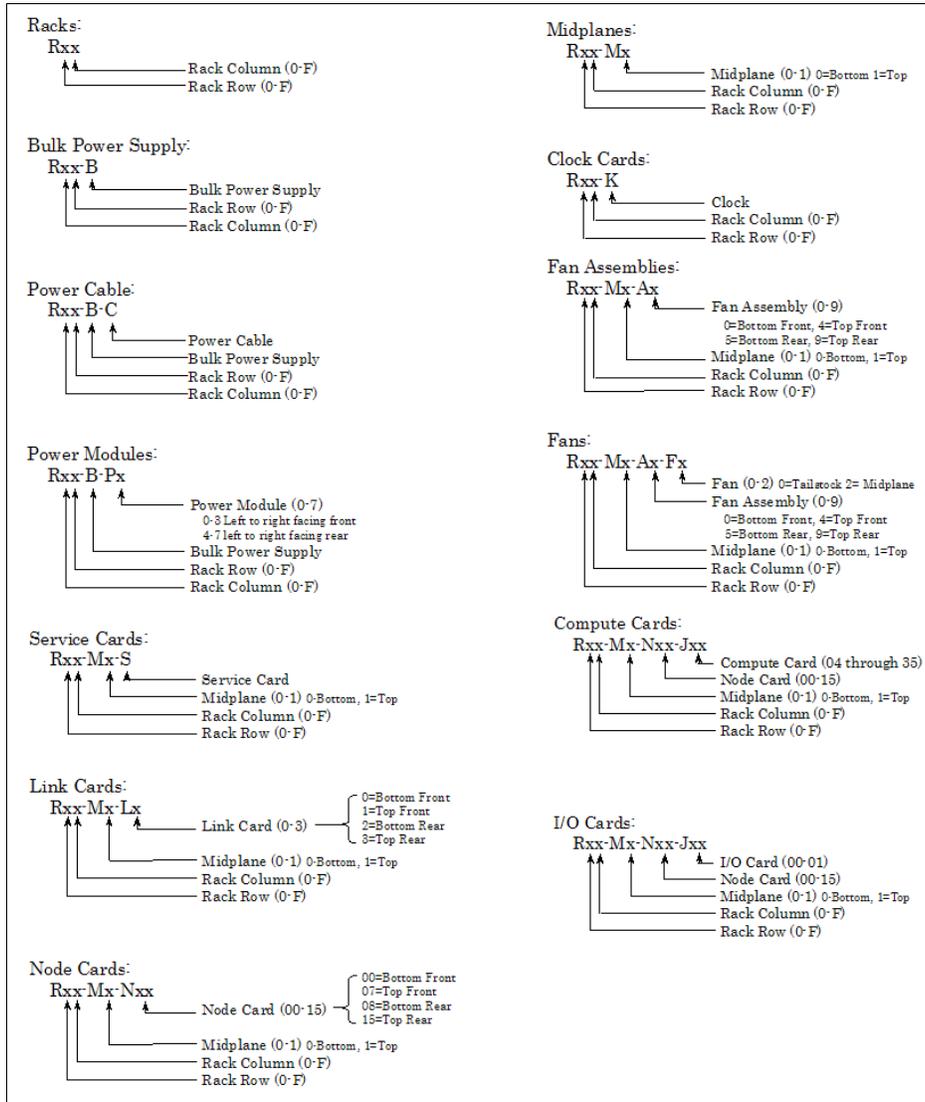


Figure 1-3 Blue Gene/P hardware naming convention

1.2.2 Database

The database has remained largely unchanged in Blue Gene/P. Figure 1-4 depicts the structure of the database on Blue Gene/P.

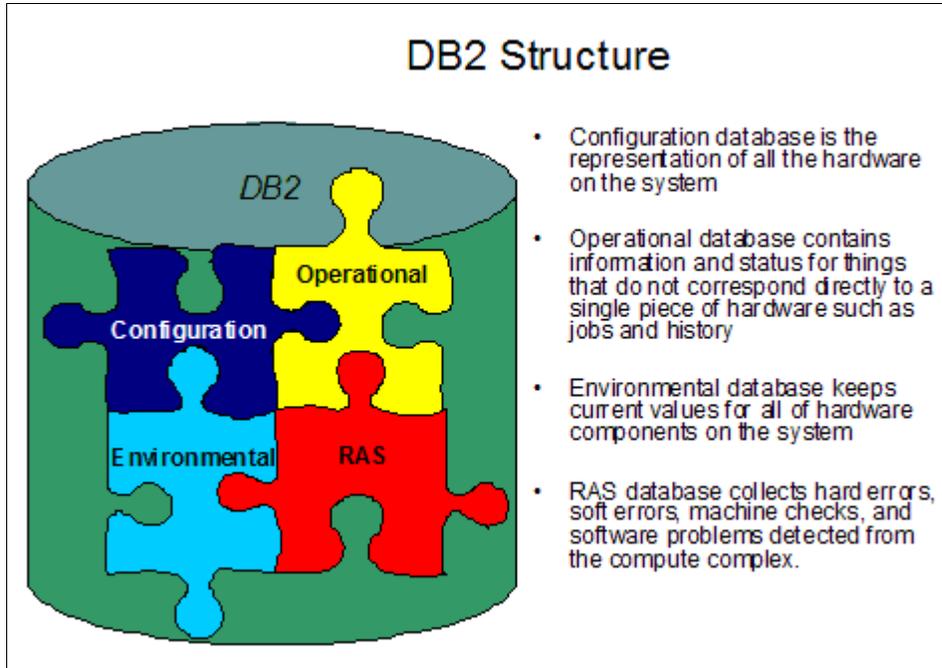


Figure 1-4 Blue Gene/P database structure

One of the more notable improvements in the database schema was the change to the primary key in all of the hardware tables. The primary key used in Blue Gene/L was the serial number, which turned out to be somewhat cumbersome. Now, with Blue Gene/P, the location string is the primary key in the tables.

With Blue Gene/L, it was possible to show more than one piece of hardware for any given location on the system. For example, if a Link card was replaced, there would actually be two link cards associated with that location. Granted, only one could be active at a time, but if you were to do some action that required the hardware to be marked missing in the database, you would not be able to distinguish which entry in the database was for hardware that is currently installed in the system.

In Blue Gene/P, that will no longer be an issue. Parts that have been replaced will still be maintained in the database but now they are stored in a history table (BGPREPLACEMENT_HISTORY table). So, from our example, the data for the currently installed Link card will be stored in the BPGLINKCARD table. The data for all of the Link cards that have been removed from the system will now reside in the BGPLINKCARD_HISTORY table.

There were some changes made to the database that stemmed from input by customers. One example of that is the BGPBLOCK table. The notion of processor sets (psets) was confusing to many. Now, rather than using the number of psets in the a block, we simply report the number of I/O nodes and compute nodes that are contained in a given block.

Database populate

After the Blue Gene/L system hardware install was completed a discovery script was run that polled all of the hardware and wrote information about that hardware back to the database.

When a Blue Gene/P is installed, after the schema has been set up, a script called `dbPopulate.pl` is run. The `dbpopulate` script requires you to provide the number of racks in the system, and how the system is configured. For example, if you have an eight rack system, you need to specify if it is configured as a one by eight (1 x 8), two by four (2 x 4), or a four by two (4 x 2). The script then fills in the database with all the possible hardware that could be on the system.

InstallServiceAction

After the `dbpopulate` script runs then the *InstallServiceAction* program starts. In short, the *InstallServiceAction* program updates the database with information about the current state of the hardware (active, missing, or in error). Service actions might be necessary to replace hardware in error or missing.

VerifyCables

The Cable Discovery scripts on Blue Gene/L actually went out to the system and found each of the cables prior to updating the database. The Blue Gene/P `dbpopulate` scripts add the cables that should exist on the system to the database. The *VerifyCables* script's function is similar to the *InstallServiceAction*. It updates the database with information about the current state of the cables.

1.2.3 Reliability, Availability, Serviceability (RAS) infrastructure

RAS has taken on a totally new format in Blue Gene/P. In Blue Gene/L, RAS was text based with no predefined format for RAS events. In Blue Gene/P, the RAS messages are very structured. There is a finite list of RAS messages that are used on Blue Gene/P, all of which have an identification (ID) number and a detailed message. The list of RAS events that have occurred or that might occur on your system are available for viewing through the Navigator. Additionally, there are 40 user-defined codes available that can be used to generate RAS messages that will be meaningful in the customer's environment.

1.2.4 Blocks

Blue Gene/P offers much more in the way of blocks, or *partitions* as they are also known. On Blue Gene/L, only small blocks with 32 or 128 nodes were supported due to the global interrupt wiring. With Blue Gene/P, you can configure any size small block (minimum of 16 compute nodes), as long as there is one I/O node contained within the block.

Small block allocation is optimized with the dynamic allocator. With the new version of the dynamic allocator the code looks at a Base Partition (BP) and if there is a node card missing the BP is marked unavailable for use with a large partition, but it will be made available for small block allocation. If an allocation attempt is made that requires resources equal to or less than the number of node cards that are available, the midplane is subdivided further. There is a minimal amount of fragmentation (orphaning) because of the optimization.

Because it is only possible to have a maximum of two I/O nodes per node card, the smallest I/O node to compute node ratio that is available is one to sixteen (1:16). Blue Gene/L offered ratios as low as one to eight (1:8) because it was possible to have as many as four I/O nodes per node card.

1.2.5 Threading

The threading implementation on Blue Gene/P supports OpenMP. The XL OpenMP implementation provides a futex compatible syscall interface so the Native POSIX Thread

Library (NPTL) pthreads implementation in glibc runs without modification. These syscalls only allow a total of four threads and provide limited support for mmap. The Compute Node Kernel provides special thread function for I/O handling in MPI.

1.2.6 Job modes

With Blue Gene/L, you had the choice of coprocessor mode or virtual node mode. Thus, your program could run on a single core in coprocessor mode, or you could split the memory and run your program on both cores in virtual node mode. With Blue Gene/P, you have three choices in this area. You can use *Symmetric Multi-Processing* (SMP) mode in which CPU 0 (MPI rank 0) runs the program's main process, as shown in Figure 1-5. The program can spawn up to three additional threads on the remaining processors.

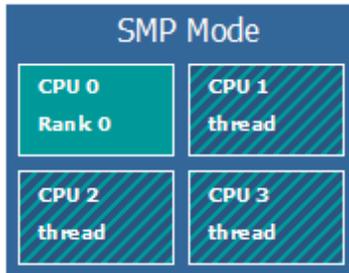


Figure 1-5 SMP mode

You can also use *Dual* mode. In Dual mode, CPUs 0 and 2 each run a main program process as shown in Figure 1-6. Each of those cores have an MPI rank and can spawn one additional thread. There is a fixed relationship between the cores. Thus, CPU 0 cannot start a thread on CPUs 1 and 3 and leave only the main process running on CPU 2. CPU 0 is only allowed send work to CPU 1, and likewise, CPU 2 can only use CPU 3 for additional threads.

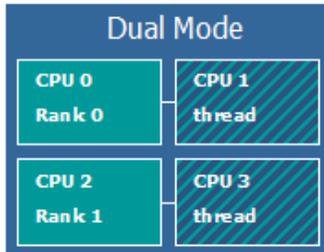


Figure 1-6 Dual processing mode

Finally, there is Virtual Node Mode (VNM), as shown in Figure 1-7. In this mode each of the cores in the processor have an MPI rank and run a program process. There is no additional threading capability in VNM.

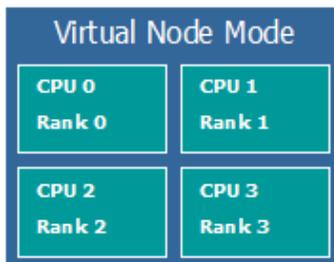


Figure 1-7 Virtual Node Mode

When submitting a High Performance Computing (HPC) job you can decide which mode your program runs in after the block has been booted. In the HPC environment the mode is a job attribute (not a block attribute). The alternative, High Throughput Computing (HTC), requires that you specify a mode when you allocate an HTC partition.

Dual Mode and Virtual Node Mode require that you split the node's memory evenly between the main processes.

1.2.7 Control System

The Blue Gene/P Control System has undergone some streamlining that makes it more efficient. When referring to the Control System (illustrated in Figure 1-8), we are generally talking about the following components:

- ▶ Low Level Control System (LLCS)
 - Machine Controller (mc)
 - mcServer
- ▶ High Level Control System, which is also known as the Midplane Management Control System (MMCS)
- ▶ DB2® (RAS interfaces)
- ▶ Scheduler APIs
- ▶ Control and I/O Daemon (CIOD)

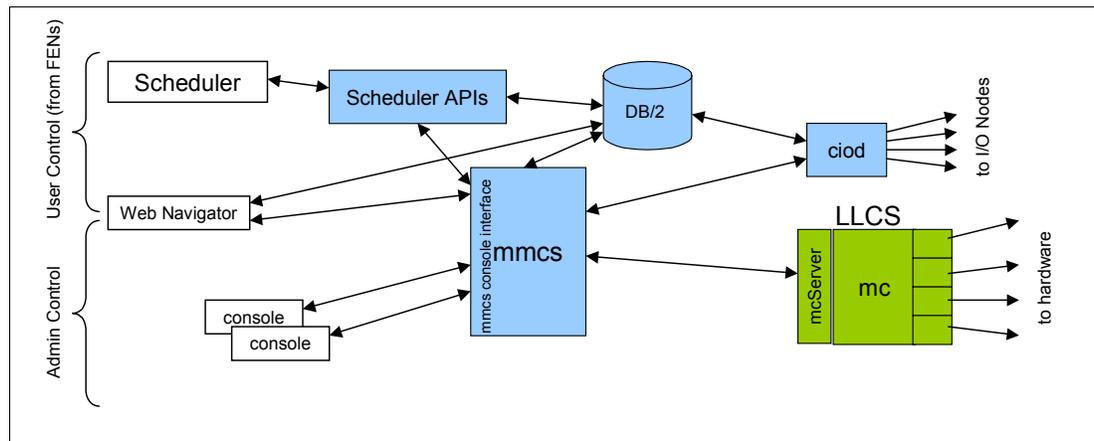


Figure 1-8 Blue Gene/P Control System

1.2.8 Proxy replaced

mcServer provides the low level access to the hardware for Blue Gene/P. This replaces the idoproxy functions that were used in Blue Gene/L. Much like the idoproxy, mcServer interfaces with the Machine Controller code to limit access to the hardware to one user at a time.

1.2.9 CIODB

Job submission requests from the various sources (schedulers, mpirun, mmcs_db_console) update the job related database tables. On Blue Gene/L there is a separate process, CIODB, that runs on the service node and handles the job loading and starting. This code polls the job table, and starts jobs, monitors jobs, and updates the job table as the job goes through the states of being loaded, started, debugged, terminated, and so forth.

This code communicates with the I/O node daemon, CIOD, running on each I/O node, to control the job's progress and monitor status.

While this processing is very similar on Blue Gene/P, there is no longer a separate CIODB process. Because the CIODB code is so closely tied to `mmcs_db_server`, the two processes have been merged and run under the `mmcs_db_server` process.

1.2.10 Midplane Management Control System

Blue Gene/L Midplane Management Control System (MMCS) binaries were originally built in 32-bit mode but later `mpirun` (and libraries it depends on) were converted to 64-bit to provide support for the 64 bit schedulers that use the `mpirun` plug-ins. MMCS in Blue Gene/P has been built with 64 bit to avoid the problems encountered with the previous version.

On Blue Gene/L, you could direct certain commands such as `write_con` to a specific node by prefixing the command with the node's index as shown by the `locate` command, such as `{0} write_con ps`. On Blue Gene/P, regular expressions can also be used as prefixes. For example, `{R00-M0-N04-J00} write_con ps` is a valid command.

The `connect` command is new, and can be used with a target specification. This can be used to boot an individual node. Some of the parameters that were on the `allocate_block` command in Blue Gene/L have been moved to the `connect` command.

There is also a new option for MMCS, `--ciod-persist`, that allows job control connections to remain open between jobs for the life of the block. This eliminates the need to create a new TCP connection to CIOD every time a job starts.

The `boot_block` command on Blue Gene/P has different parameters from Blue Gene/L. You can specify the microloader, compute node and I/O node elf images on 'boot_block' rather than issuing individual load commands.

Blue Gene/P also includes the ability to perform multiple connect-boot-disconnect sequences without reallocating the block. You can disconnect from the block's hardware resources without freeing the block and reconnect without reallocating the block. This feature allows you to boot the block several times without freeing it.

Reconnecting blocks and jobs

If CIODB ended for some reason while a job was running on a Blue Gene/L system, both the job and the block would stay active, but the ability to monitor and manage the job would be lost. Similarly, on Blue Gene/P if the `mmcs_db_server` server stops, the block stays booted, and the job keeps running on the nodes. With Blue Gene/P, you can allow reconnects when the server is restarted. This functionality is enabled simply by using the `--reconnect` option when starting the `mmcs_db_server`. When the server is restarted, the job can continue to write to `stdout` and `stderr`.

Note: Remember that any of the application's output that was sent to `stderr` or `stdout` while the server was down will be missing.

1.2.11 Hardware monitoring

Blue Gene/L had a separate process called *Hardware Monitor*. In Blue Gene/L, there were concerns about the amount of resources that were required to monitor the hardware. There were also several methods of starting the monitor functions on Blue Gene/L. On Blue Gene/P, the monitoring functions are integrated with the control system. The Environmental

Monitor starts and ends with `mmcs_db_server`. The impact on the system as a whole is reduced because there are only three connections to `mcServer` to collect the data at a time. The Environmental Monitor comes with the polling interval set to five minutes.

With the original Hardware Monitor, there were two different graphical user interfaces (GUIs) that could be used to view the data collected (Navigator and VNC). The results gathered by the Monitor in Blue Gene/P can only be accessed through Blue Gene Navigator. Figure 1-9 shows the initial page that displays when Environmental Queries is selected in the Navigator.

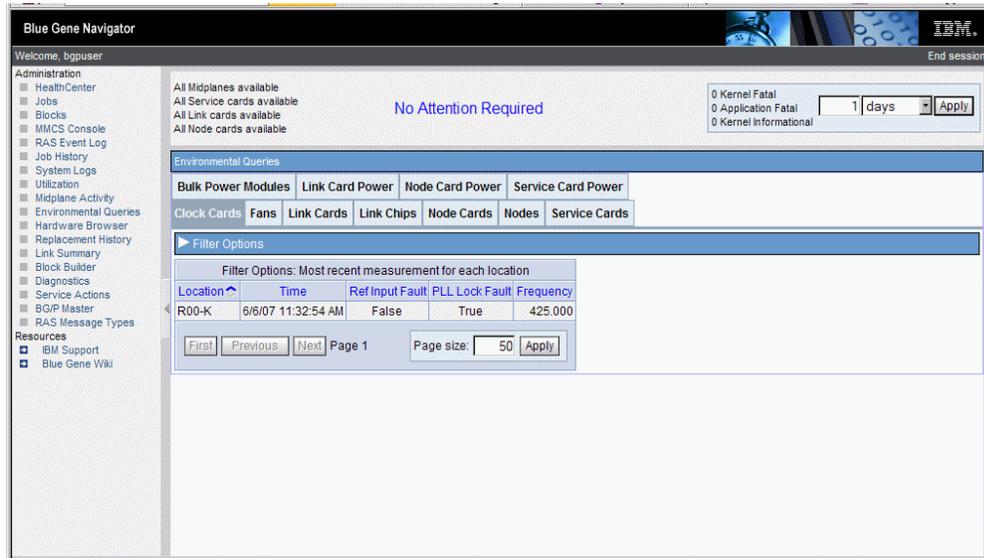


Figure 1-9 Environmental queries

In the Blue Gene/L version of the monitor, there was no limit on the length of time that data collected was stored. Monitoring and controlling the amount of data that was kept was entirely up to the administrators of the system. By default, Blue Gene/P purges the data every three months. The configuration can be altered to store more or less data as required by the local environment.

1.2.12 Client/server mpirun

The client/server `mpirun` on Blue Gene/P remains largely unchanged from Blue Gene/L. The most notable exception is the removal of the `rsh/ssh` mechanism for initiating the back-end process. Using the `rsh` or `ssh` protocols required that each user have a profile on the Service Node. In Blue Gene/P, a daemon process running on the service node handles connections from front-end `mpirun` processes and fork back-end (`mpirun_be`) `mpirun` processes.

Figure 1-10 is a diagram of how `mpirun` interacts with the remainder of the control system. After `mpirun_be` is forked, the sequence of events for booting partitions, starting jobs, and collecting `stdout/stderr` is very similar to Blue Gene/L `mpirun`.

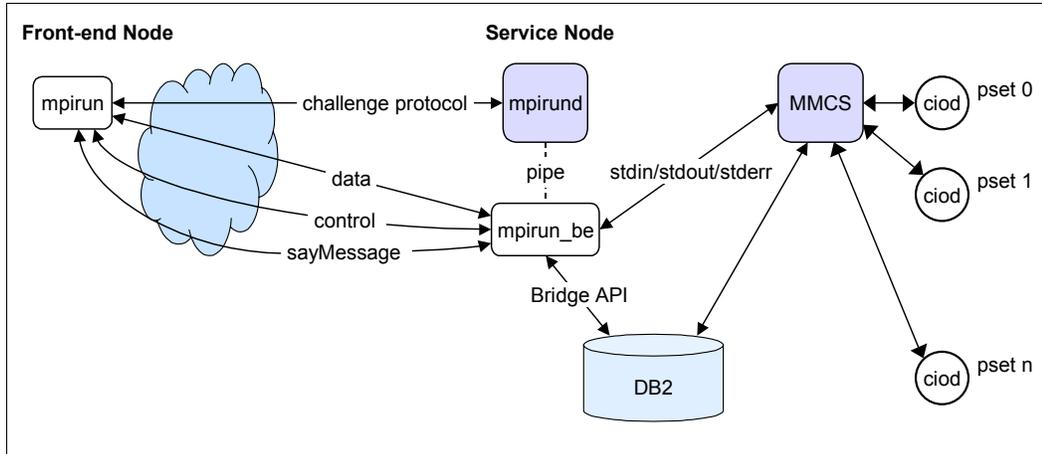


Figure 1-10 The mpirun flow

Another change in the Blue Gene/P version of mpirun is the support for Multiple Program Multiple Data (MPMD) style jobs. With MPMD a different executable, arguments, environment, and current working directory can be supplied for a single job on a processor set (pset) basis. For example, a user can execute four different executables on a partition that contains four psets. This function is handled by a new tool called *mpiexec*.

Note: The *mpiexec* tool should not be confused with the *mpiexec* style of submitting a Single Program Multiple Data (SPMD) parallel MPI job.

1.2.13 Bridge APIs

In Blue Gene/L, the Control System stored temporary XML files in the /tmp directory. This could cause issues occasionally if the /tmp file system became full. Blue Gene/P resolves this issue by no longer writing XML files to /tmp. Instead, they are passed in-memory. This change has improved overall Control System performance as well.

Users of the Bridge APIs (for example, mpirun and LoadLeveler®) should see significant performance improvements because of the implementation of database connection pooling and a reduction in the number of database connections required on some Bridge APIs.

1.2.14 Navigator updates

The original purpose of Blue Gene Navigator was to provide administrators with a GUI from which they could manage their system. Since that time, the Navigator has also evolved into a user tool that shows the health of the system, its utilization, and the availability of system resources. This section highlights some of the additional improvements made to the Navigator in Blue Gene/P.

Replacement History

One of the new items added to the Navigator is the Replacement History link. This link allows you to view all of the various hardware types that have been replaced and filter your view based on specific times, location, serial number, or electronic chip ID (ECID).

Block Builder

The Block Builder feature introduced in the Blue Gene/L version of the Navigator allowed users to create blocks that consisted of one or more midplanes. In Blue Gene/L, the more

complicated blocks such as sub-midplane (small) blocks needed to be defined using XML. Blue Gene/P has eliminated the need for the customers to do any XML. All of the necessary functionality is provided in the Navigator and the Bridge APIs. This includes small blocks and blocks doing pass-through.

Service Actions

With Blue Gene/L, you could query the Service Actions table to view current or previous actions that have been performed on the system. Blue Gene/P allows administrators to actually initiate a Service Action from the Navigator.

BGPMaster

An interface to manage BGPMaster was added to the Navigator. You can start, restart, or stop the mcservers, mmcs_server, mpirund, navigator_server, submit_server and realtime_server from the GUI.

Status messages concerning each of the servers are displayed. For example, if a server has been stopped, the message shows what action is required to recover and how long the server has been down.

RAS Message Types

A link was added to the Navigator that allows you to search on the types of messages that have occurred on your system. The default query brings up all message types that have occurred in the last day (24 hours), but you can filter to specific message IDs or time periods.

Plug-ins

The ability to change some of the information that is displayed by Blue Gene Navigator was added in Blue Gene/P. Now, you can write plug-ins to customize your interface. You can add to or replace the existing graphs at the bottom of the page. You can also add additional links to the navigation pane. By default, the Attention area at the top of the page alerts you when certain events occur, such as hardware failures or RAS events. You can customize this area to notify you of events that can have significance in your environment.

1.2.15 Parts replacement

Performing service actions on Blue Gene/P will not have the same impact that it did on Blue Gene/L. Originally, even changing a compute node required administrators to take an entire midplane out of service for a short time. In Blue Gene/P, the service actions are more finite. They only affect the specific hardware that is being serviced. A good example of that is the same procedure that replaces a compute node. Now, this process only affects partitions (and jobs running on them) that include the specific node card that contains that compute node rather than the whole midplane. An even better example is the service action that is performed to replace a Link card. On a multiple rack system, this type of service would require that all the racks in the same row, and all the racks in the column, be put into service mode. The Blue Gene/P Service Actions no longer need to power down all of the neighboring link cards. This results in a dramatic difference in the amount of time it takes to prepare for and end the service action.

The Blue Gene Navigator has an interface that allows you to perform service actions and to also see which jobs and blocks will be impacted by the service action that you are going to perform.

1.2.16 Diagnostics

Diagnostics on Blue Gene/P are similar to the set of tests that were available on Blue Gene/L with the addition of test cases added to exercise new features such as DMA.

Still initiated from Blue Gene Navigator, you can track the progress and cancel a diagnostic run if necessary. Results are stored in the database and viewable from the Navigator. There are cross-reference links between RAS messages, failures, and hardware locations.

You can select between small, medium, or large sets of tests plus an additional test bucket added that is labelled *complete*. The complete option includes all the tests in the large suite, plus a single node Linpack and a torus connectivity test.

Diagnostics are more efficient in Blue Gene/P because pipelining has been introduced into the harness. The harness runs tests on the hardware while compiling results from previously run tests.

RAS and error reporting is improved for both diagnostics and normal system usage. For example, RAS events are now used to indicate failures as opposed to error messages being sprinkled throughout test output. This event-driven model provides many benefits:

- ▶ Reduced time spent parsing code
- ▶ Improved diagnostics runtime
- ▶ Decreased network utilization

The new RAS subsystem provides not only low-level error reporting but decoding facilities in the control system to enhance error messages on the way back up to the user. Diagnostics also have a total view of all the RAS events posted during a test and has the ability to make decisions based on all the information presented.



2

Blue Gene Navigator

This chapter provides an overview of the Blue Gene Navigator. We discuss some of the features of the Navigator and give instructions on how to install and modify the interface.

2.1 Blue Gene Navigator overview

Blue Gene Navigator is designed to be a full featured graphical user interface (GUI) that supports the administration of the Blue Gene/P core. You can also configure Navigator to be a user-oriented interface that is useful in providing information about usage, jobs, and system availability to those who might be submitting jobs to the Blue Gene/P core. Navigator is installed with the system software. The administrative view is the default set of pages that are enabled when the system is installed.

BGPMaster controls the Tomcat HTTP server that Navigator runs on. By default, the instance requires that Secure Sockets Layer (SSL) be configured and the server listens on port 32072. Administrators will access the page with the URL

`https://hostname:32072/BlueGeneNavigator`. The Navigator runs with the profile of the user that starts BGPMaster. In most environments, this profile is the `bgpadmin` user. The user that starts BGPMaster must have read access to the `/etc/shadow` files to allow the Navigator to perform authentications.

The supported browsers for Blue Gene Navigator are Firefox 2.0 and Internet Explorer® 7.0 or later.

2.1.1 Setup

The installation of the Blue Gene/P system covers the initial setup of the Navigator. However, if the occasion arises that requires the system be reinstalled there are a few steps that need to be completed prior to starting the Navigator.

Setup script

The DB2 libraries must be made available to Tomcat so that it can access the database, and the JAAS plugin for interfacing to Linux PAM must be made available so that Tomcat can authenticate users. A script is provided to do the setup. Example 2-1 shows the syntax for running the script.

Example 2-1 Run the setup script

```
$ cd /bgsys/drivers/ppcfloor/navigator
$ ./setup-tomcat.sh
```

By default, the setup shown in Example 2-1, configures the Navigator to allow only authenticated users to access the Web interface. To enable anonymous access to user pages, you need to copy the `web-withenduser.xml` file into Navigator's configuration, as shown in Example 2-2.

Example 2-2 Set up anonymous user pages

```
$ cd /bgsys/drivers/ppcfloor/navigator
$ cp web-withenduser.xml catalina_base/webapps/BlueGeneNavigator/WEB-INF/web.xml
```

Navigator uses PAM to authenticate users. It uses the bluegene PAM stack so this must be set up by creating a file `/etc/pam.d/bluegene`. Example 2-3 shows a sample of the `/etc/pam.d/bluegene` file.

Example 2-3 The `/etc/pam.d/bluegene` file

```

#%PAM-1.0
auth    include      common-auth
auth    required     pam_nologin.so
account include     common-account
password include    common-password
session include     common-session

```

Note: The Navigator runs as whoever starts up BGPMaster and that user is the one who performs the authentications. So, for example, if `bgpadmin` starts up BGPMaster and `/etc/pam.d/bluegene` is configured to use `pam_unix2.so`, then `bgpadmin` must be able to read the `/etc/shadow` file. If this is not the case, no one will be able to sign on to Navigator.

SSL Configuration

By default, the Navigator uses the `/bgsys/local/etc/navigator.keystore` keystore. This must be created when the system is configured. To do this, run the `keytool` command as shown in Example 2-4. Replace `myhostname.example.com` with your service node's host name. The file should be owned by `bgpadmin` group.

Example 2-4 The `keytool` command

```

$ keytool -genkey -alias tomcat -keyalg RSA -keystore
/bgsys/local/etc/navigator.keystore
Enter keystore password: changeit
What is your first and last name?
[Unknown]: myhostname.example.com
What is the name of your organizational unit?
[Unknown]: STG
What is the name of your organization?
[Unknown]: IBM
What is the name of your City or Locality?
[Unknown]: Rochester
What is the name of your State or Province?
[Unknown]: MN
What is the two-letter country code for this unit?
[Unknown]: US
Is CN=myhostname.example.com, OU=STG, O=IBM, L=Rochester, ST=MN, C=US correct?
(type "yes" or "no")
[no]: yes

Enter key password for <tomcat>
(RETURN if same as keystore password):  changeit

$ chgrp bgpadmin /bgsys/local/etc/navigator.keystore

$ chmod 660 /bgsys/local/etc/navigator.keystore

```

The Navigator is configured to accept only SSL connections, because when using PAM authentication it needs access to the clear text password so that the browser cannot adequately mask the password it sends without using SSL. The Navigator can be configured

to accept non-SSL connections by editing the file `/bgsys/drivers/ppcfloor/navigator/catalina_base/conf/server.xml` and removing the XML comment characters from the line with `<Connector port="32071" />`. The non-ssl port is 32071.

Block Visualizer

The Blue Gene Navigator administrative web application has a new feature that allows visualization of blocks in the Block Details view. This is enabled by installing the JavaView applet to the Navigator web application directory. The steps for installing the JavaView applet are as follows:

1. Download the zip file containing the JavaView applet from the JavaView web site at <http://www.javaview.de>. Click on *Download*, then select *javaview.zip* to download the zip file.
2. Unzip *javaview.zip* in a temporary directory using the command:


```
bgpadmin@YourSys:/tmp> unzip javaview.zip
```
3. Copy the file `jars/javaview.jar` to `/bgsys/drivers/ppcfloor/navigator/catalina_base/webapps/BlueGeneNavigator/resources/` directory.
4. Make sure this file is readable by `bgpadmin`. The Navigator must be restarted if it is running when the file is copied into the directory.


```
bgpadmin@YourSys>/bgsys/drivers/ppcfloor/bin/bgpmaster restart
navigator_server
```

Authority

There are three roles defined in Navigator: End User, Service, and Administrator. The Linux group to Navigator role mapping is defined in the local Navigator configuration file. The Administrator group has access to all pages. Table 2-1 describes the pages that Navigator allows users in the End User and Service roles.

Table 2-1 Group authorizations

Navigator Link	Page	End-user access	Service access
	about.jsp		X
BG/P Master	bgMaster.jsp		
Blocks	blocks.jsp	X	X
	blockDetails.jsp	X	X
Block Builder	blockbuilder.jsp		
Diagnostics	diagnostics.jsp		X
	configDiags.jsp		X
	diagsBlock.jsp		X
	diagsLocation.jsp		X
	diagsLocTest.jsp		X
	diagsLog.jsp		X
	diagsRun.jsp		X
	diagsTestcase.jsp		X

Navigator Link	Page	End-user access	Service access
Environmental Queries	environmentals.jsp		X
RAS Message Types	errCodes.jsp	X	X
Hardware Browser	hardwareBrowser.jsp		X
Replacement History	hardwareReplacement.jsp		X
Health Center	healthcenter.jsp	X	X
Jobs	jobs.jsp	X	X
	jobDetails.jsp	X	X
Job History	jobHistory.jsp	X	X
Link Summary	linkSummary.jsp		X
System Logs	systemLogs.jsp		X
	logFileView.jsp		X
Midplane Activity	midplaneActivity.jsp	X	X
MMCS Console	mmcs.jsp		
	mmcsblockoutput.jsp		
Service Actions	serviceActions.jsp		X
	prepareServiceAction.jsp		
RAS Event Log	ras.jsp	X	X
	rasDetails.jsp	X	X
Utilization	utilization.jsp		

2.1.2 Navigator options

This section describes options that are available to configure the Navigator to better suit your environment.

Startup options

Navigator is started from BGP Master, which calls the `/bgsys/local/etc/startnavigator` script. BGP Master passes in several arguments to `startnavigator`. `startnavigator` uses `baselogdir`, `useDatabase`, and `dbproperties` such as the other startup scripts to log the output. `startnavigator` sets the `CATALINA_BASE` environment variable to `<navigatorpath>/catalina_base`. When `catalina.sh` is started, it uses this as the directory containing the tomcat instance files.

Several of the `startnavigator` options are turned into arguments such as `-Dcom.ibm.bluegene.binpath=<binpath>` and passed on to `catalina.sh`, that starts Tomcat in which Navigator runs. Then, Navigator grabs them when it starts up and uses them for its own configuration.

--binpath Path to control system binaries, defaults to current directory, usually `/bgsys/drivers/ppcfloor/bin`. Navigator calls `<binpath>/bgpmaster` when a user goes to the BG/P Master page.

--configpath	Path to configuration files, defaults to /bgsys/local/etc. Navigator looks for the local Navigator configuration file (navigator-config.xml) here and bgpmaster.init (used on the BG/P Master page).
--dbproperties	db.properties file, defaults to <configpath>/db.properties. If this is set, Navigator reads this for the DB properties, otherwise it would use <configpath>/db.properties.
--db2profile	DB2 profile, defaults to db2profile, but usually ~bgsysdb/sqllib/db2profile. startnavigator sources this file.
--useDatabase	Machine SN, defaults to BGP. Navigator uses this to construct the default logs directory on the System Logs page.
--baseLogdir	Path to logs, defaults to /bgsys/logs.
--navigatorpath	Path to Navigator files, defaults to <binpath>/navigator. Navigator looks for the floor navigator configuration file under this directory navigator-config.xml.

Options in the Navigator configuration file

The Navigator configuration files contain several options. The floor Navigator configuration file (usually /bgsys/driver/ppcfloor/navigator/navigator-config.xml) contains the default values. The default values can be overridden by putting the same option in your local navigator configuration file (/bgsys/local/etc/navigator-config.xml) with the new value.

You must create a local Navigator configuration file for authentication if nothing else. A sample local Navigator configuration file is shipped in /bgsys/drivers/ppcfloor/navigator/sample-local-navigator-config.xml. This file should be copied to /bgsys/local/etc and edited with the administrator, service, and user groups that are defined in your local configuration.

Administrator groups

Users in these Linux groups have access to all the sections in the Navigator. To have multiple groups, add a <administrator-group>groupname</administrator-group> for each group. Note that the value can be a Linux group name or gid (Example 2-5).

Example 2-5 Administrator configuration file entry

```
<administrator-group>bgpadmin</administrator-group>
```

Service groups

Users in these Linux groups have access to the service sections in the Navigator. To have multiple groups, add a <service-group>groupname</service-group> for each group. Note that the value can be a Linux group name or gid (Example 2-6).

Example 2-6 Service configuration file entry

```
<service-group>bgpservice</service-group>
```

User groups

Users in these Linux groups have access to only the end-user pages in the Navigator. These pages do not allow any updates to the Blue Gene system. To have multiple groups, add a <user-group>groupname</user-group> for each group. Note that the value can be a Linux group name or gid (Example 2-7).

Example 2-7 User group configuration file entry

```
<user-group>bgpuser</user-group>
```

Default chart timeout

This is the timeout, in minutes, for the charts that are displayed in the Dashboard when the chart configuration does not set a timeout (Example 2-8).

Example 2-8 Chart timeout value

```
<default-chart-timeout>10</default-chart-timeout>
```

Plugin library path

The plugin library path provides the directory that the Navigator will look in to find plugin libraries (jars), as shown in Example 2-9.

Example 2-9 Plugin library path entry

```
<plugin-library-path>/bgsys/local/lib</plugin-library-path>
```

Environmental options

The Environmental options contain two parameters that are related to highlighting the temperature readings on the Environmentals page, minimum and maximum temperature values. Any temperature reading below the min-temp is blue, while everything above the max-temp is red, with shades of blue, green, and red for values between the minimum and maximum (see Example 2-10).

Example 2-10 Environmental limits

```
<environmentals>
  <min-temp>25.0</min-temp>
  <max-temp>50.0</max-temp>
</environmentals>
```

BGPMaster options

These options provide the client executable and port used to call bgpmaster. If the client executable does not start with a slash (/), then the <binpath> is prepended. A value of default for the port means that the default port for BGPMaster is used. See Example 2-11.

Example 2-11 BGPMaster options

```
<bg-master>
  <client-exe>bgpmaster</client-exe>
  <port>default</port>
</bg-master>
```

MMCS Console options

The Navigator's MMCS console interface provides nearly the same functionality that is available when you access the console from a shell session (see Example 2-12). The Navigator provides a few features that are either not available or appear differently than the command line version. For a more complete listing of functions, see 2.3.4, "MMCS Console" on page 39.

The configurable options are:

- ▶ Maximum Reply Size: MMCS messages larger than this value will be truncated.

- ▶ **Maximum History Size:** This is the maximum size of all the MMCS messages that display on the MMCS page. Replies are removed from the back of the history after this size is reached.
- ▶ **Redirect Applet Log Level:** The log level for the Redirect Applet. This affects what goes into the Java™ console on the user's browser. The values are the different severity levels in java.logging.
- ▶ **mmcs_db_server Options:** This is the host name and the port that Navigator uses to connect to the mmcs_db_server.
- ▶ **Redirect DB Server Options:** This is the host name and the port that the Navigator uses to connect to the redirect server.

Example 2-12 MMCS Console options

```
<mmcs-console>
  <max-reply-size>8029</max-reply-size>
  <max-history-size>2097152</max-history-size>
  <redirect-applet-log-level>INFO</redirect-applet-log-level>
  <mmcs-db-server>
    <hostname>127.0.0.1</hostname>
    <port>32031</port>
  </mmcs-db-server>
  <redirect-server>
    <hostname>127.0.0.1</hostname>
    <port>32032</port>
  </redirect-server>
</mmcs-console>
```

2.2 Common functions

There are many features in the Blue Gene Navigator that remain common throughout the entire interface. Although most of the features are intuitive, in this section we highlight these features and provide comments on their use. We start by accessing the administrative version of Navigator using the URL <https://ServiceNodeName:32072/BlueGeneNavigator> in your browser of choice to open the Jobs page shown in Figure 2-1. This page is the default home page.

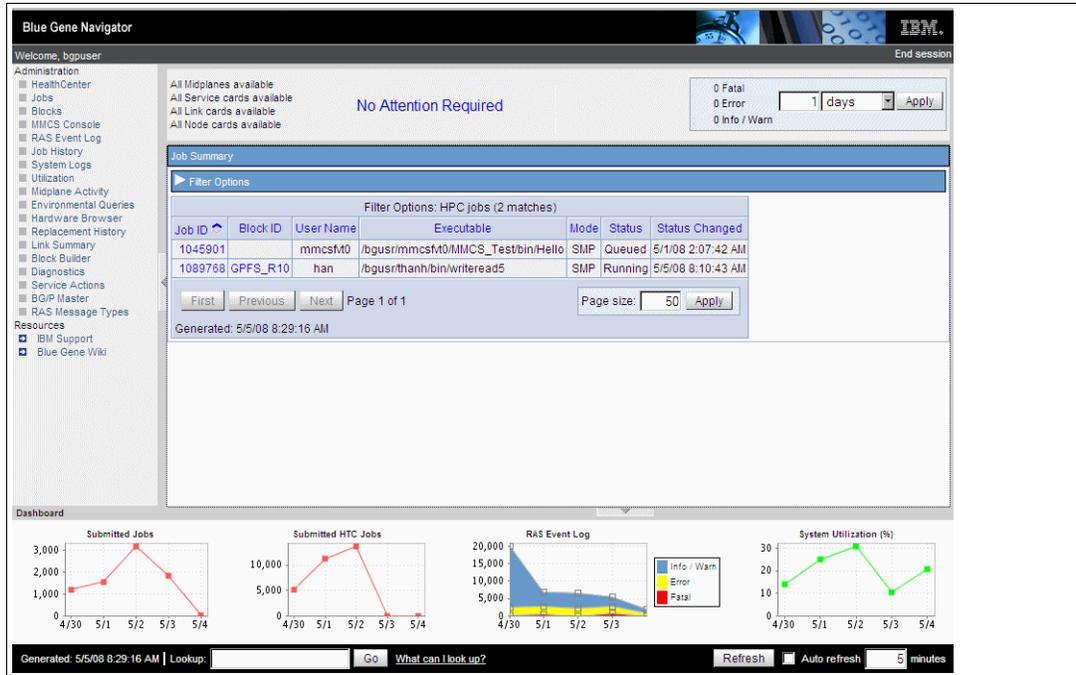


Figure 2-1 Jobs page

Note: Figure 2-1 is shown with the High Throughput Computing jobs graph. By default that is not configured on the system. Instructions for adding the chart are located in the section labelled “High Throughput Computing (HTC) Jobs graph” on page 33

At the top, left of the page you can see the user ID that is currently logged in and on the right there is a link to close the session (as shown in Figure 2-2).



Figure 2-2 Top bar

Just below the title bar that shows the current user is the *Attention area* (Figure 2-3). On the left side of this area, the current status of the hardware in the Blue Gene/P core displays. If there are any issues, the number and type of hardware that is not available is reported in this section. On the right side of this area, you can query the database for Kernel and Application errors based on time. Supply a number in the open selection box (default is 1 day) and then using the drop-down menu and select either seconds, minutes, hours, or days. Click **Apply** to run the query, and the results are returned and posted within the shaded box. Notice that as long as there are no errors reported, the *No Attention Required* message is displayed in the center of this section.

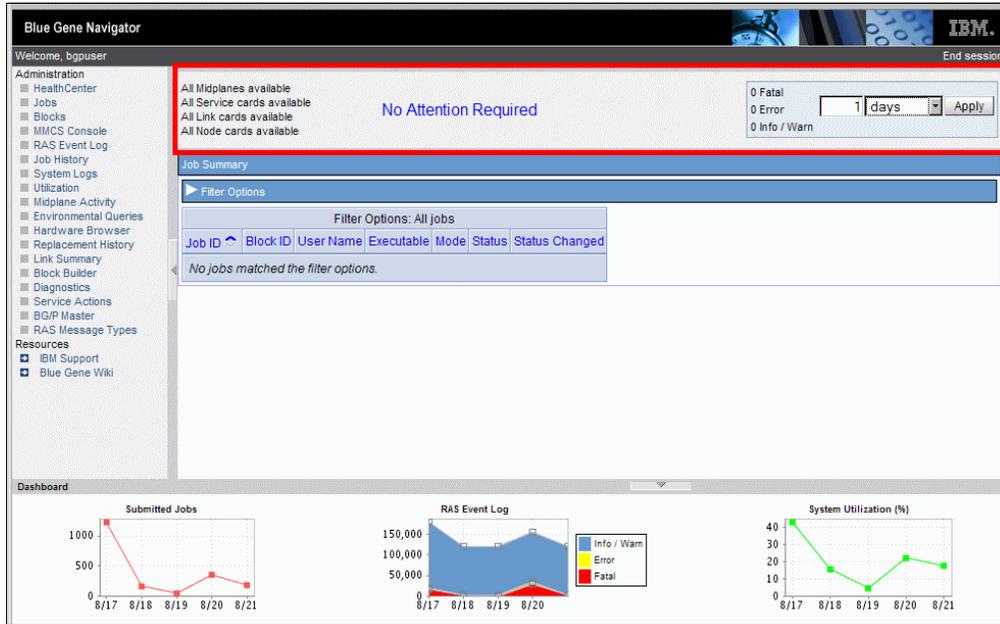


Figure 2-3 Attention area

If there was a problem the message in the center of the area changes to that shown in Figure 2-4. There is a default set of events that trigger the attention message that displays. You can customize the attention area of the Navigator to alert you based on events of your choice (2.4, "Customizing Blue Gene Navigator" describes how to do this). Clicking **Attention** takes you to the Health Center page so that you can locate the source of the problem.

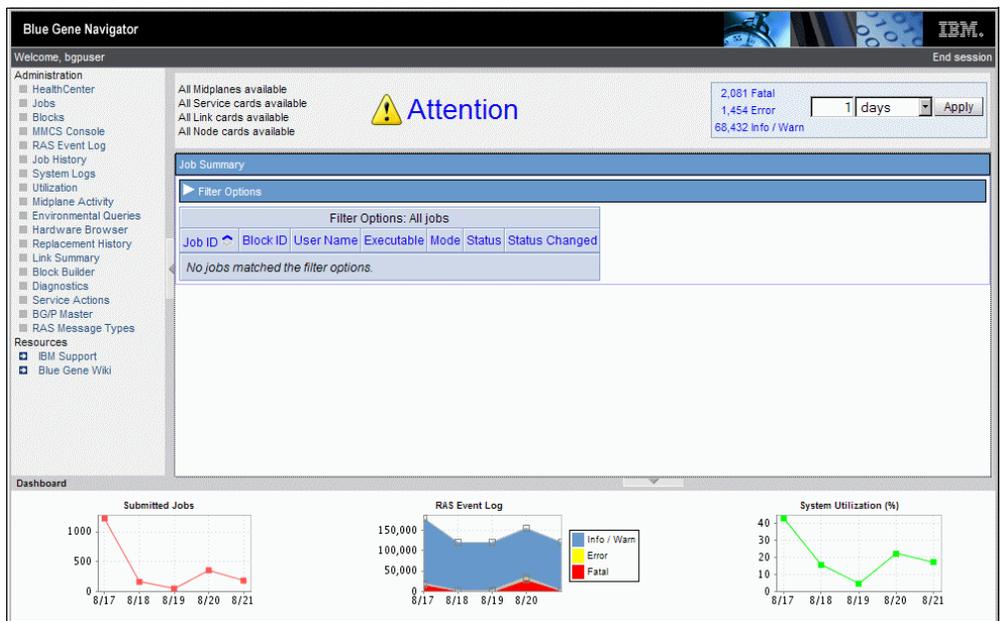


Figure 2-4 Attention required message

In the event that hardware has become unavailable, you see a summary listed in the Attention area as shown in Figure 2-5. Clicking any of the affected hardware takes you to the Health Center page. From there, you can identify the individual pieces of hardware that are not available.

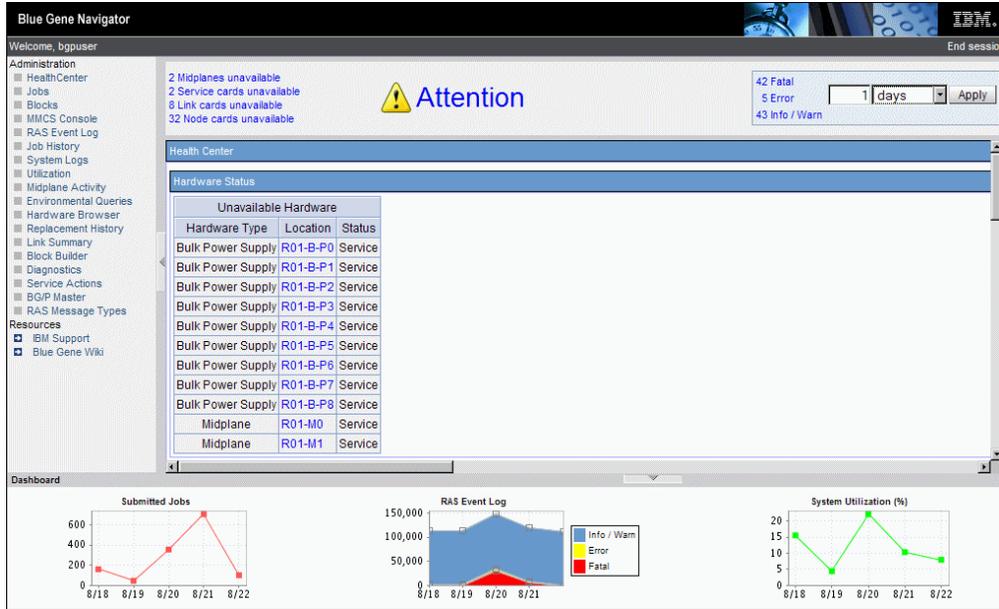


Figure 2-5 Unavailable hardware

On the extreme left of the panel is the *navigation pane* shown in Figure 2-6. Clicking any of the links under the Administration heading takes you to another page that is designed to administer or query that specific subject. You can find details about each of the links in 2.3, "Using the Navigator" on page 35. You can also customize the lower section of the navigation pane. You can add links that might be useful in your environment. By default, only the IBM Support link is provided.

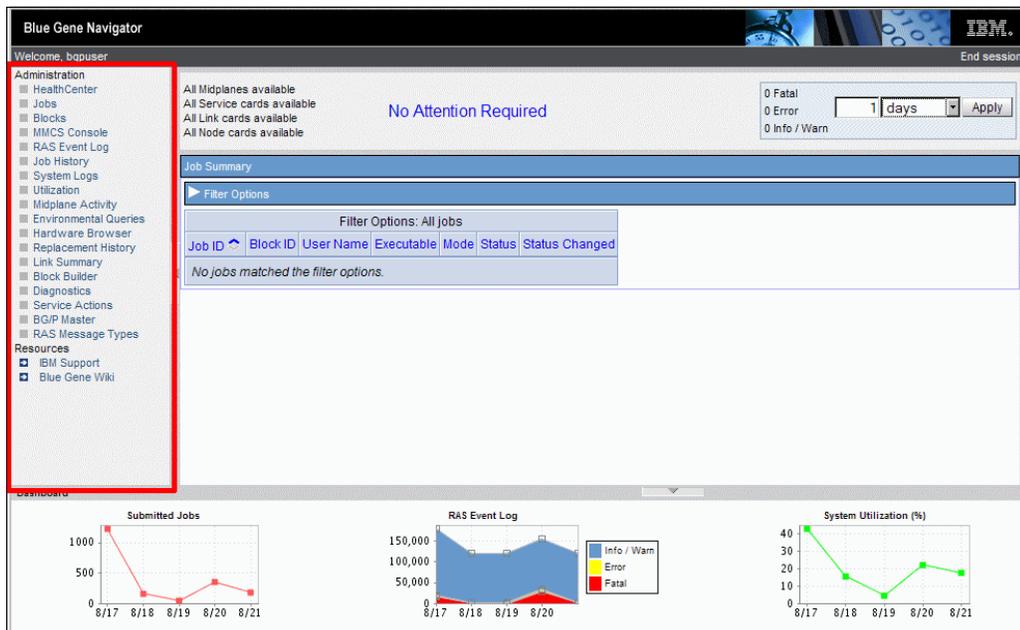


Figure 2-6 Navigation pane

After clicking a link in the navigation pane the content is loaded in the center frame. For consistency, we refer to this as the *content area* (Figure 2-7). There are several features in the content area that are available in most views.

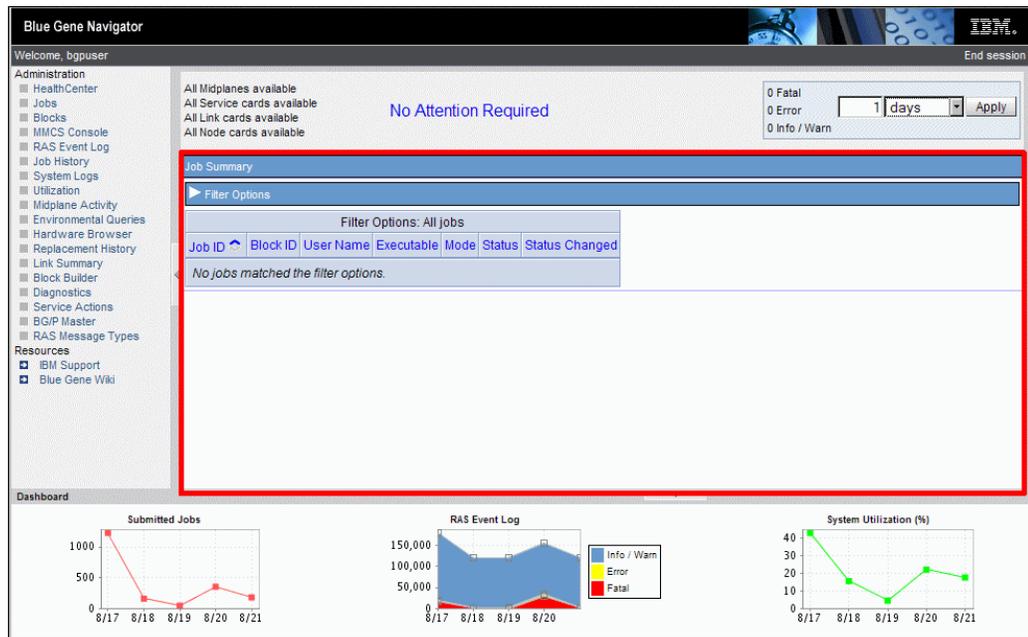


Figure 2-7 Content area

Almost every selection that you can make in the navigation page opens a page that has a Filter Options link. Clicking the white triangle (sometimes referred to as a *twisty*, shown in Figure 2-8) link opens a display that gives you the available filter options for that panel.



Figure 2-8 Twisty icon

Figure 2-9 shows the Filter Options that are available for the Job History page.

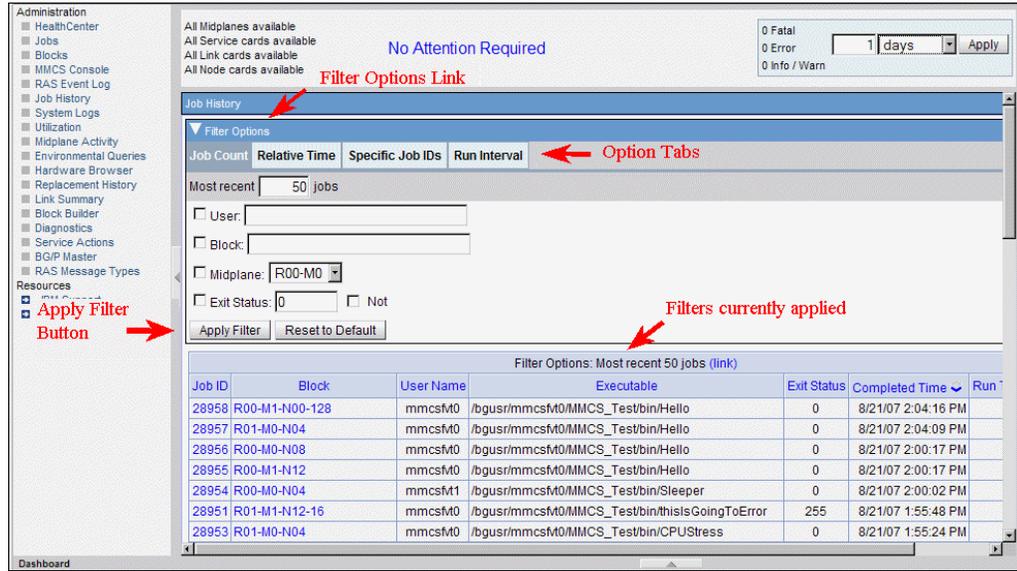


Figure 2-9 Filter Options

Notice on this page, just below the Filter Options title bar, that there are four option tabs available:

- ▶ Job Count
- ▶ Relative Time
- ▶ Specific Job IDs
- ▶ Run Interval

Clicking any one of these tabs gives a new set of options that you can use to filter the data returned. The number of tabs, and the options available on the tabs, are different in the various environments.

After you have narrowed the search by supplying the filters, select **Apply Filter** located at the bottom of the options to submit a query using your selected filters. When a filter is applied on a given page, it is stored in a session cookie. Every time you come back to that page, the same filter is applied unless you click **Reset to Default** or choose new filters.

Just below the Filter Options section of the page there is a title bar that contains the current filters. After the list of options the (link) hotspot allows you to save the selected filter options in your browser's bookmarks. To do this, right-click the hotspot and select the option to bookmark the page. Selecting the saved bookmark reruns the query and returns the most current data that matches your filters.

Blue text in the Navigator indicates that it is a link to more information or provides a separate function. For example, clicking the column headings when they are blue sorts the data in the content portion of the page. In Figure 2-10, we sorted by clicking the *Location* column head. You can see the up arrow (^) is blue, indicating that the data is sorted in ascending order by location. Clicking the Location heading a second time will sort the data in descending order.

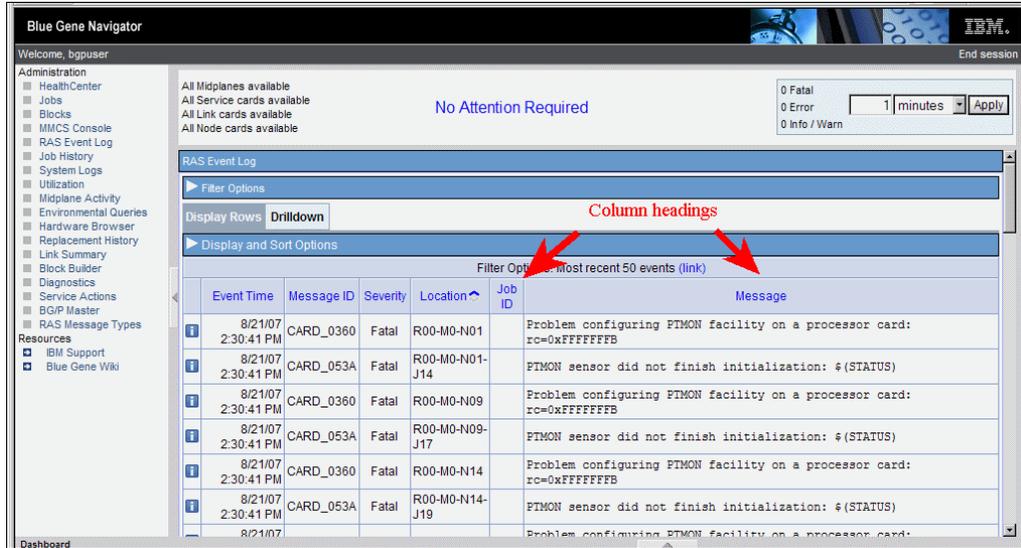


Figure 2-10 Click the Column head to sort

In the extreme left column of the RAS event log, there is an additional information icon shown in Figure 2-11.



Figure 2-11 Additional information icon

Clicking the icon displays a message details page such as the one shown in Figure 2-12. The message details contain information about the error and, in some cases, what action, if any, should be taken.

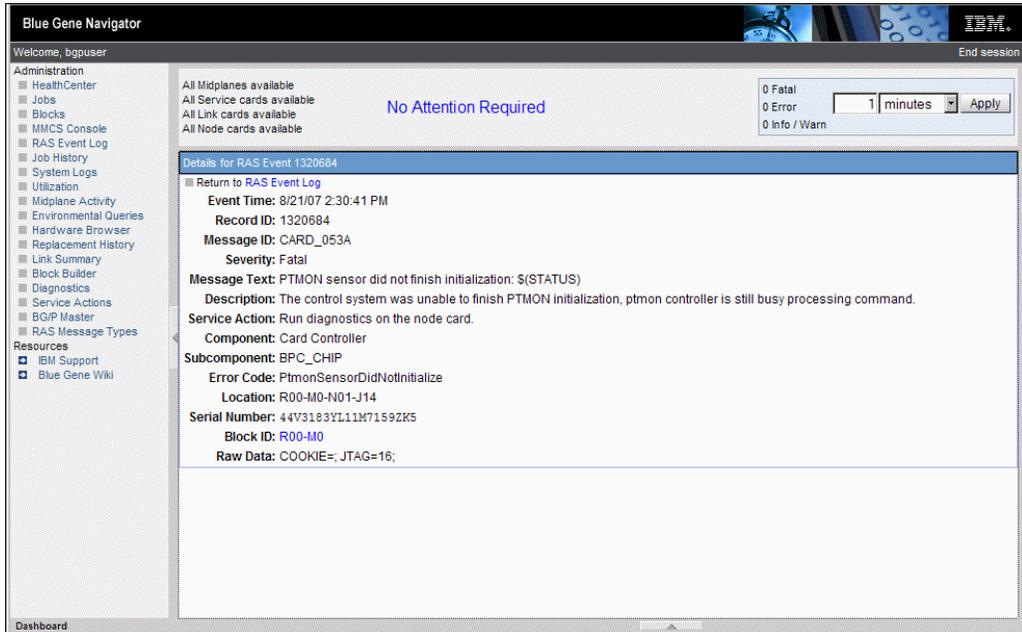


Figure 2-12 RAS event detail page

The bottom section of the Navigator pages, Figure 2-13, is referred to as the *Dashboard*. By default, the Dashboard section of the Navigator contains three graphs. The content of the graphs give you a feel for the system's utilization and general health.

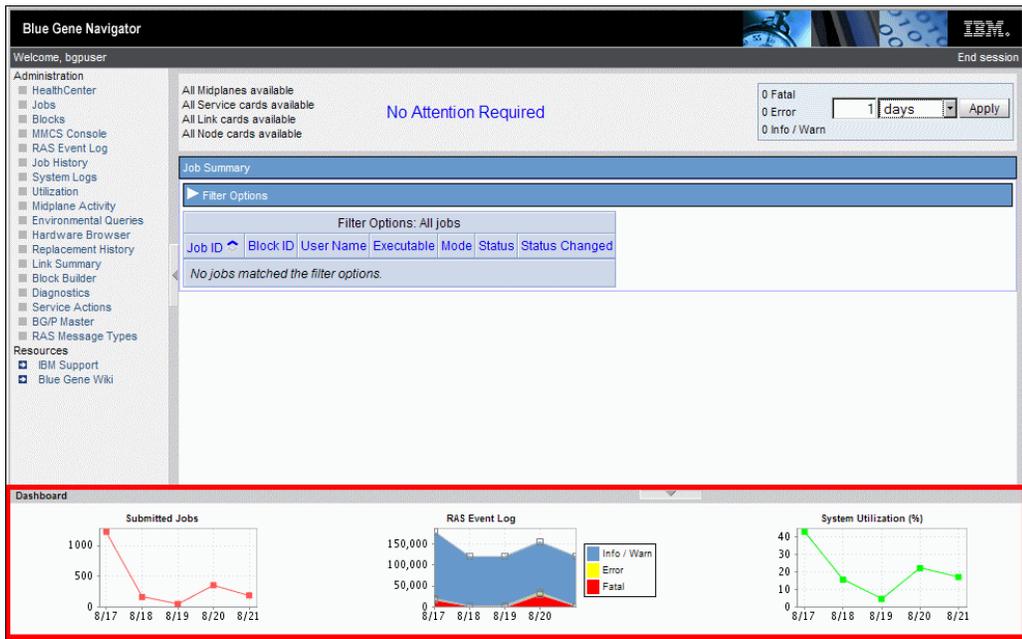


Figure 2-13 Navigator Dashboard

You can collapse the Dashboard by clicking the down arrow on the splitter bar (Figure 2-14). Clicking the arrow again restores the Dashboard.



Figure 2-14 Splitter bar

Submitted Jobs chart

The Submitted Jobs chart (Figure 2-15) shows a summary of the jobs that have been submitted over the last five days. Hovering over any of the red squares in the graph gives you the date and the total of jobs counted on that day. Clicking anywhere on the chart takes you to the Job History page. You can find more details on the Job History page in 2.3.6, “Job History” on page 43.

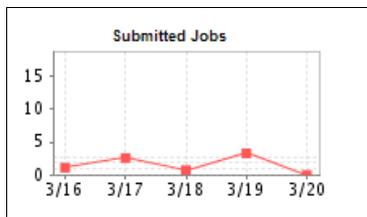


Figure 2-15 Submitted Jobs graph

RAS event graph

The Software RAS Events chart (Figure 2-16) gives you a color-coded, visual report of RAS events for the last five days. When you hover over any of the squares, you see a recap of the date and number of that particular type of error. Clicking any of the squares submits a query on that type of error, and you are directed to the RAS Event Log page. You can find more information about that log in 2.3.5, “RAS Event Log” on page 41.

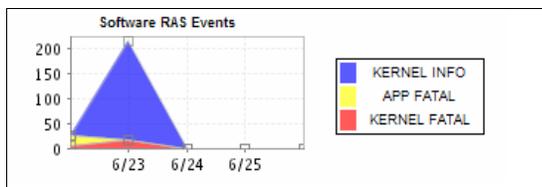


Figure 2-16 RAS graph

System Utilization graph

Finally, the System Utilization graph (Figure 2-17) shows the percentage of utilization for the date specified. As with the other charts, when you hover over one of the squares, you see the daily results. For further details about system Utilization, see 2.3.8, “Utilization” on page 46.

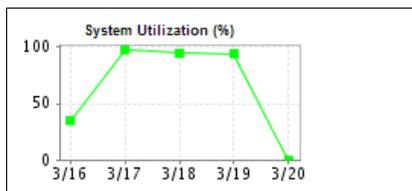


Figure 2-17 System Utilization graph

The three graph view of the Dashboard applies to the user version of the Navigator also. The only differences being that the mouse-over function does not display any data on the System Utilization graph nor does the link to the Utilization page work.

High Throughput Computing (HTC) Jobs graph

The Navigator Dashboard can be customized to show submitted HTC jobs as well as HPC type jobs. The code to generate the graph is included in the release. To enable the function you will have to add the chart-plugin container to the local Navigator configuration file (/bgsys/local/etc/navigator-config.xml). The bold lines in Example 2-13 need to be inserted into the file.

Example 2-13

```
<navigator>
. . .
  <chart-plugin>
    <name>htc_jobs</name>
    <enabled>true</enabled>
  </chart-plugin>
. . .
</navigator>
```

Save the file and then restart the Navigator using the following command (as bgpadmin):

```
./bgpmaster restart navigator_server
```

Figure 2-18 shows the Dashboard with the new graph.

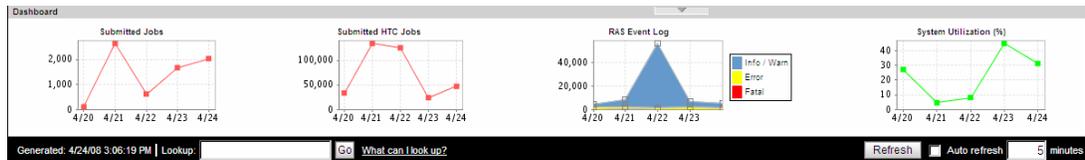


Figure 2-18 Navigator Dashboard with HTC graph

The Dashboard is another area of the Navigator that can be redefined by you, as described in 2.4, “Customizing Blue Gene Navigator” on page 57

Lookup tool

The bottom section of the Navigator interface contains a *Lookup* tool and an *Auto refresh* function. The Lookup tool allows you to search for block or job information. You can search on specific information or use the Lookup to browse to other pages within the Navigator. Clicking on the *What can I look up?* link will bring up the tool’s help page.



Figure 2-19 Search bar

To the right side of the search bar is the refresh function. The *Refresh* button located on this bar is the recommended method of refreshing your page. You can also set your browser to automatically refresh by supplying the value (in minutes) that you want to wait before renewing the display and then clicking in the check box to start. Once you have enabled the Auto refresh it will continue to refresh on any Navigator page you have open.

2.2.1 User preferences and session data

Blue Gene Navigator stores user preferences by user name. If several system administrators log in to Blue Gene Navigator with the same user name, they all share the same user preferences. Therefore, we recommend that every system administrator use a different user name when logging in to Blue Gene Navigator.

Table 2-2 shows the user preferences saved by user name.

Table 2-2 User preferences saved from session to session

Item	Description
AttentionMagnitude	The interval for the RAS events in the Attention area
AttentionUnit	The type of units (seconds, minutes, hours, days) to use when filtering RAS events shown in the attention area
AutoHighlight	Auto highlight enabled or disabled in Log Viewer
DashboardDisplayed	Whether the Dashboard is displayed or not
DiagsLogQueryPageSize	Number of rows on a page for the Diagnostics results by location
DiagsRunQueryPageSize	Number of rows on a page for the Diagnostics results by diagnostics run
Directory	Default directory for log files in System Logs
ErrcodesIntervalUnit	Interval for Error Codes event count filter option
ErrcodesIntervalMagnitude	Interval for Error Codes event count filter option
ErrcodesPageSize	Number of rows on a page for the Error Codes results
JobHistoryJobType	Job Type filter option in Job History
JobHistoryPageSize	Number of rows on a page for the Job History results
JobSummaryJobType	Job Type filter option in Job Summary
JobSummaryPageSize	Number of rows on a page for the Job Summary results
Lines	Number of lines to display in Log Viewer
RASEventLogPageSize	Number of rows on a page for the RAS events results
RefreshTimeout	Refresh time for page auto-refresh
ReplHistPageSize	Number of rows on a page for the Replacement History results
Timer	Auto-refresh interval in Log Viewer

Blue Gene Navigator also maintains session data, such as whether the Dashboard is currently visible at the bottom of the Web page (see Figure 2-13 on page 31). If you open multiple tabs or multiple windows in the current browser session, all tabs and windows share the same session data.

2.3 Using the Navigator

Using the Blue Gene Navigator is very intuitive. Select a link in the navigation pane and you are taken to a page that allows you to accomplish tasks that are associated with that link. This section introduces the features that are available in the Navigator, many of which you use in the day-to-day administration of the Blue Gene/P system.

2.3.1 Health Center

We touched briefly on the Health Center in the 2.1, “Blue Gene Navigator overview” on page 18. In this section, we take a closer look at the Health Center and explore its functionality.

Figure 2-20 shows the Health Center main page.

The screenshot shows the Blue Gene Navigator interface. The top bar includes the title 'Blue Gene Navigator', a user name 'Welcome, bguser', and an 'End session' button. The left navigation pane lists various system administration tasks. The main content area features a warning message: 'Attention' with a yellow triangle icon. Below this, the 'Hardware Status' section shows a table of 'Unavailable Hardware' with columns for Hardware Type, Location, and Status. The 'Recent Jobs with Fatal RAS Events' section displays a table with columns for Job ID, Block, User Name, and Executable.

Hardware Type	Location	Status
Bulk Power Supply	R01-B-P0	Error
Bulk Power Supply	R01-B-P2	Error
Bulk Power Supply	R01-B-P4	Error

Job ID	Block	User Name	Executable
28932	R00-R01	vany	umt2k.070814
28929	R00-R01	vany	umt2k.070814
28900	R01-M0-N04-128	Klig	/bgusr/mklight/run_flash3_shockyc/_acceptance/flash3
28859	R01-M0-N00	ubbod	/bgusr/subbodda/extmpibm/extmpibm.rts
28812	R01-M0-N00	ubbod	/bgusr/subbodda/extmpibm/extmpibm.rts
28811	R01-M0-N00	ubbod	/bgusr/subbodda/extmpibm/extmpibm.rts
28705	R01-M1	hhuan	/bgusr/qhhuang/POKMPISYS/DRV320_2007-070814P-SLES10-DD2/bin/mpi_32_8
28645	R01	ayeru	/bgusr/jayeruva/BGTH_BGP/test1024nDD2BGP/pallas/pall1024DD2SMP/bgpsftsn-R01/MB-MPI1.64KB.perf
28637	R01	ayeru	/bgusr/jayeruva/BGTH_BGP/test1024nDD2BGP/pallas/pall1024DD2SMP/bgpsftsn-R01/MB-MPI1.64KB.perf
28623	R01	ayeru	/bgusr/jayeruva/BGTH_BGP/test1024nDD2BGP/pallas/pall1024DD2SMP/bgpsftsn-R01/MB-MPI1.4MB.perf

Figure 2-20 Health Center home

In the content area of the main page, there are the following subheadings:

- ▶ Hardware Status
- ▶ Recent Jobs with Fatal RAS Events
- ▶ Running Jobs with Proactive Power Management
- ▶ Running Jobs with Reactive Power Management
- ▶ Recent Fatal RAS Events
- ▶ Attention Messages

Note: The Attention Messages subheading is dynamic. It displays only in the Health Center if you have customized the Attention plugin.

Hardware status

When there are no current hardware issues with the Blue Gene/P hardware, this message box simply returns the message *All hardware is available*. However, if there is hardware that is missing or somehow in error, the Navigator shows a message similar to the one shown in Figure 2-21, which supplies the hardware type, location, and status.

Hardware Status		
Hardware Type	Location	Status
Bulk Power Supply	R00-B-P0	Missing
Bulk Power Supply	R00-B-P1	Missing
Bulk Power Supply	R00-B-P2	Missing
Bulk Power Supply	R00-B-P3	Missing
Bulk Power Supply	R00-B-P4	Missing
Bulk Power Supply	R00-B-P5	Missing
Bulk Power Supply	R00-B-P6	Missing
Bulk Power Supply	R00-B-P7	Missing
Bulk Power Supply	R00-B-P8	Missing
Clock Card	R00-K	Missing
Fan Module	R00-M0-A5	Missing
Fan Module	R00-M0-A6	Missing
Fan Module	R00-M0-A7	Missing
Fan Module	R00-M0-A8	Missing
Fan Module	R00-M0-A9	Missing
Link Card	R00-M0-L0	Missing
Link Card	R00-M0-L1	Missing

Figure 2-21 Health Center hardware status window

Recent jobs with fatal RAS events

This section of the Health Center reports all of the jobs that ran in the previous 24 hours (default) that reported an RAS event. Figure 2-22 shows a the report includes information about, and a link to, the job and block. The table also shows the user, executable, time of, and the message that is associated with the event.

Recent Jobs with Fatal RAS Events			
Job ID	Block	User Name	Executable
29509	R01-M1	subb	/bgusr/subbodda/BGTH_BGP/test512nDD2BGP/extmpibm/extmpibm512DD2/bgpsftsn-R01-M1/extmpibm.rts
29601	R01-M0-N04-128	subb	/bgusr/subbodda/BGTH_BGP/test128nDD2BGP/extmpibm/extmpibm128DD2/bgpsftsn-R01-M0-N04-128/extmpibm
29591	R01-M0-N00	subb	/bgusr/subbodda/BGTH_BGP/test32nDD2BGP/extmpibm/extmpibm32DD2/bgpsftsn-R01-M0-N00/extmpibm.rts
29587	R01-M0-N00	subb	/bgusr/subbodda/BGTH_BGP/test32nDD2BGP/extmpibm/extmpibm32DD2/bgpsftsn-R01-M0-N00/extmpibm.rts
28932	R00-R01	ivan	umt2k.070814
28929	R00-R01	ivan	umt2k.070814
28900	R01-M0-N04-128	mkli	/bgusr/mklight/run_flash3_shockcyl_acceptance/flash3
28859	R01-M0-N00	subb	/bgusr/subbodda/extmpibm/extmpibm.rts
28812	R01-M0-N00	subb	/bgusr/subbodda/extmpibm/extmpibm.rts
28811	R01-M0-N00	subb	/bgusr/subbodda/extmpibm/extmpibm.rts
28705	R01-M1	qhh	/bgusr/qhhuang/POKMPISYS/DRV320_2007-070814P-SLES10-DD2/bin/mpi_32_8
28645	R01	jaye	/bgusr/jayeruva/BGTH_BGP/test1024nDD2BGP/pallas/pall1024DD2SMP/bgpsftsn-R01/MB-MPI1.64KB.perf.rts
28637	R01	jaye	/bgusr/jayeruva/BGTH_BGP/test1024nDD2BGP/pallas/pall1024DD2SMP/bgpsftsn-R01/MB-MPI1.64KB.perf.rts
28623	R01	jaye	/bgusr/jayeruva/BGTH_BGP/test1024nDD2BGP/pallas/pall1024DD2SMP/bgpsftsn-R01/MB-MPI1.4MB.perf.rts
28611	R01	jaye	/bgusr/jayeruva/BGTH_BGP/test1024nDD2BGP/pallas/pall1024DD2DUAL/bgpsftsn-R01/MB-MPI1.16KB.perf.rts
28322	R01-M1-N08-128	mkli	/bgusr/mklight/run_flash3_shockcyl_acceptance/flash3
28321	R01-M1-N04-128	mkli	/bgusr/mklight/run_flash3_shockcyl_acceptance/flash3
28301	R01-M1-N04-128	mkli	/bgusr/mklight/run_flash3_shockcyl_acceptance/flash3
28300	R01-M1-N08-128	mkl	/bgusr/mklight/run_flash3_shockcyl_acceptance/flash3

Figure 2-22 Jobs with RAS events

To drill down to the actual RAS event, click the job number in the far left column to open the Job Details page shown in Figure 2-23. At the bottom of the page there is a table that contains the number of each type RAS message generated by this job.

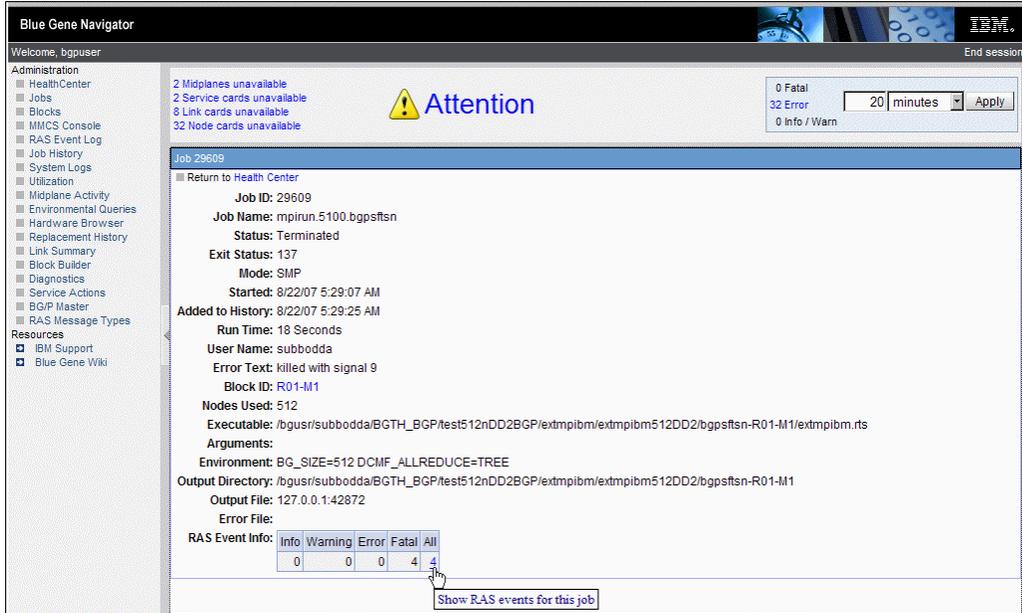


Figure 2-23 Health Center Job details page

The last column of the table, labelled *All*, contains a link to the RAS database. Clicking the number in the cell below *All* sends a query to the database and returns all the RAS events for the job (Figure 2-24).

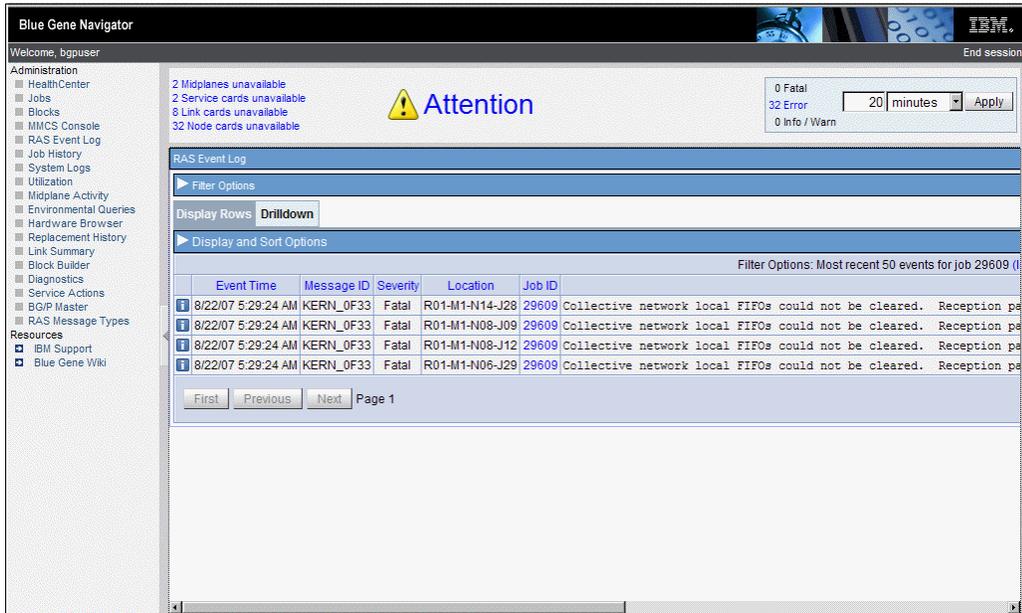


Figure 2-24 RAS query results

Recent fatal RAS events

This section of the Health Center reports the time, location, and description of fatal errors that have occurred on the system (Figure 2-25). The *Block* column provides a link to the block that contains the hardware that reported the error.

Recent Fatal RAS Events					
Time	Block	Job ID	Location	Message	
8/22/07 12:25:13 PM	R00-M1		R00-M1-N00	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFFB	
8/22/07 12:25:13 PM	R00-M1		R00-M1-N00-J32	PTMON sensor did not finish initialization: \$(STATUS)	
8/22/07 12:25:13 PM	R00-M1		R00-M1-N01	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFFB	
8/22/07 12:25:12 PM	R00-M1		R00-M1-N01-J09	PTMON sensor did not finish initialization: \$(STATUS)	
8/22/07 12:25:12 PM	R00-M1		R00-M1-N10	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFFB	
8/22/07 12:25:12 PM	R00-M1		R00-M1-N10-J23	PTMON sensor did not finish initialization: \$(STATUS)	
8/22/07 12:25:11 PM	R00-M1		R00-M1-N09	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFFB	
8/22/07 12:25:11 PM	R00-M1		R00-M1-N09-J26	PTMON sensor did not finish initialization: \$(STATUS)	
8/22/07 12:25:11 PM	R00-M1		R00-M1-N04	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFFB	
8/22/07 12:25:11 PM	R00-M1		R00-M1-N04-J28	PTMON sensor did not finish initialization: \$(STATUS)	
8/22/07 12:25:09 PM	R00-M1		R00-M1-N14	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFFB	
8/22/07 12:25:09 PM	R00-M1		R00-M1-N14-J32	PTMON sensor did not finish initialization: \$(STATUS)	
8/22/07 12:25:09 PM	R00-M1		R00-M1-N13	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFFB	
8/22/07 12:25:09 PM	R00-M1		R00-M1-N13-J20	PTMON sensor did not finish initialization: \$(STATUS)	
8/22/07 12:25:08 PM	R00-M1		R00-M1-N01	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFFB	
8/22/07 12:25:08 PM	R00-M1		R00-M1-N01-J35	PTMON sensor did not finish initialization: \$(STATUS)	
8/22/07 12:25:08 PM	R00-M1		R00-M1-N07	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFFB	
8/22/07 12:25:08 PM	R00-M1		R00-M1-N07-J35	PTMON sensor did not finish initialization: \$(STATUS)	
8/22/07 11:30:51 AM	DefaultControlEventListener		R00-M1-S	Successfully powered off this card.	
8/22/07 11:30:51 AM	DefaultControlEventListener		R00-M1-N15	Successfully powered off this card.	

Figure 2-25 Fatal errors reported in HealthCenter

2.3.2 Jobs

The Jobs link shows you a summary of jobs that are currently running or in the queue to be run. With the introduction of HTC, there may be thousands of jobs in the Blue Gene database. This can be overwhelming for the user and take a long time to display, so the Navigator allows for many filtering options and paging of the running jobs. Figure 2-26 shows all of the options available to filter your view of jobs currently running on the system.

Job Summary

Filter Options

Job type: Both HPC and HTC HPC-only HTC-only

Status: Attach Debugger Begin Dying Start Error Debug Loaded Load Queued Running Ready to Start Terminated

Mode: DUAL SMP VN

Block ID:

User Name:

Executable:

HTC Pool:

HTC location:

Service Location:

Filter Options: HPC jobs (4 matches)						
Job ID	Block ID	User Name	Executable	Mode	Status	Status Changed
1204398		mmcsfM0	/bglhome/millerjc/TWIST_WORK/bin/Hello	SMP	Queued	5/27/08 11:24:30 AM
1209635		mmcsfM0	/bglhome/millerjc/TWIST_WORK/bin/Hello	SMP	Queued	5/29/08 12:15:07 PM
1209819	R00-M0-N07-J01_test	mklight	/bgusr/mklight/Expersist_test.cna_cpp.elf	DUAL	Dying	5/29/08 3:22:47 PM
1209825	R00-M0-N10-J00	blocksom	.toolkit/armcl/src/perf_base.x	SMP	Ready to Start	5/29/08 3:22:47 PM

First Previous Next Page 1 of 1

Page size:

Generated: 5/29/08 3:22:49 PM

Figure 2-26 Navigator Jobs page

2.3.3 Blocks

The Blocks section of the Navigator gives you the option of viewing blocks that are *In Use*, *Available*, or *Free* (Figure 2-27). You can see the blocks in any one of the categories by clicking the appropriate tab. By default, the page contains all blocks that fit the description on the tab, on all midplanes in your system. The Filter Options allows you to narrow the results that display in each section to a single midplane, mode or by HTC pool.

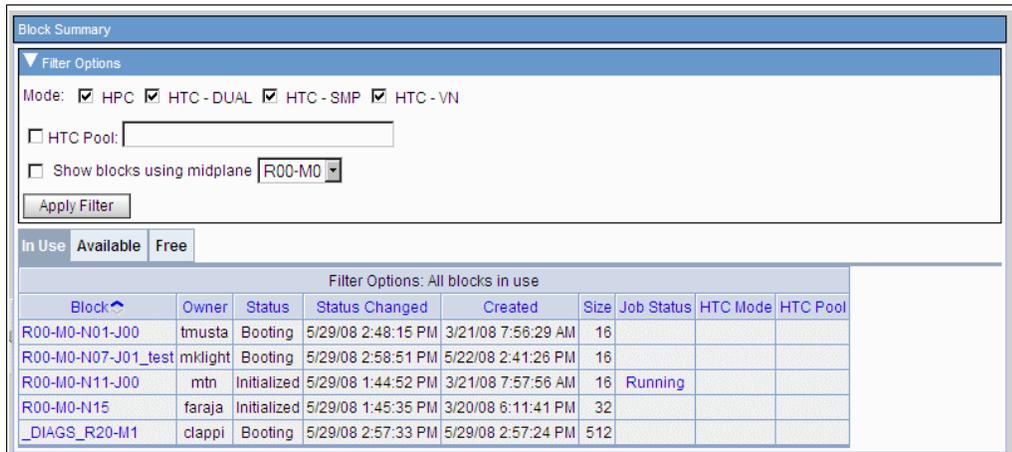


Figure 2-27 Block Summary page

2.3.4 MMCS Console

Figure 2-28 shows the Blue Gene Navigator version of the Midplane Management Control System (MMCS) console. The MMCS console portion of Blue Gene Navigator provides nearly the same functionality as the mmcs_db_console shell. The interface supplies a text box to enter the MMCS commands. The response is returned in the window below the command line.

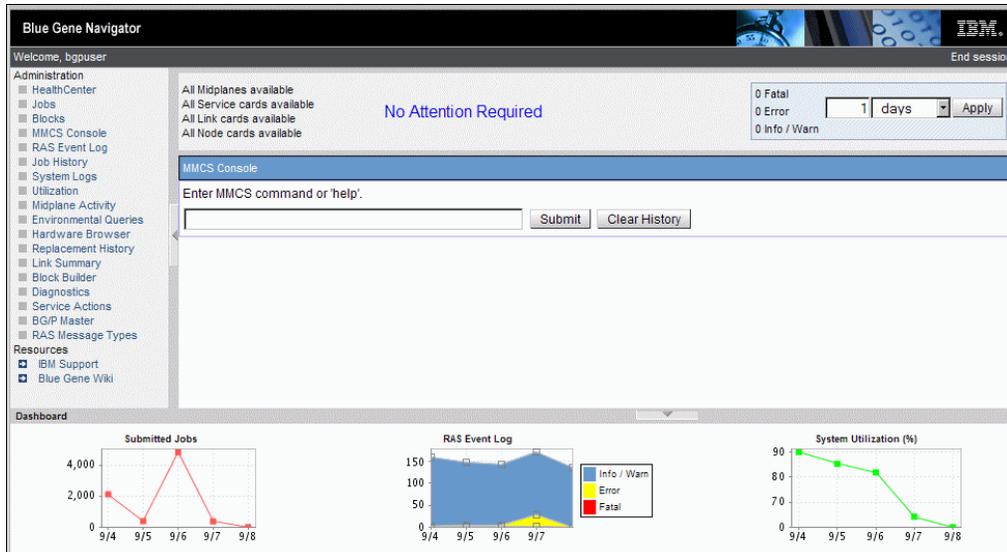


Figure 2-28 MMCS console

Each command's output is printed at the bottom of the panel and is associated with its own arrow, status line, and command-recycle icon. Figure 2-29 provides a legend of the icons that you see on the MMCS console page.

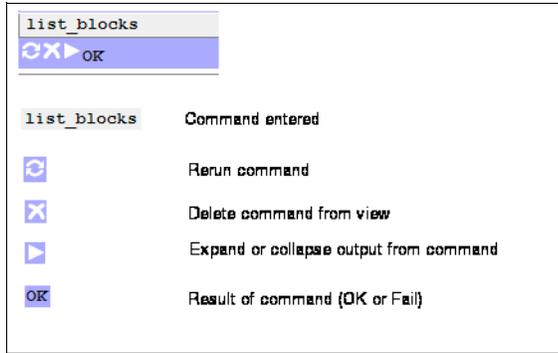


Figure 2-29 MMCS console legend

In this example, we ran several `list` commands followed by a `getblockinfo` command. You can use the scroll bar, to the right of the main content area, to see the commands run and to see the output during the current session. The Navigator displays the most recent command and its output at the bottom of the panel. When a new command is executed the output from the previous request is collapsed. See Figure 2-30.

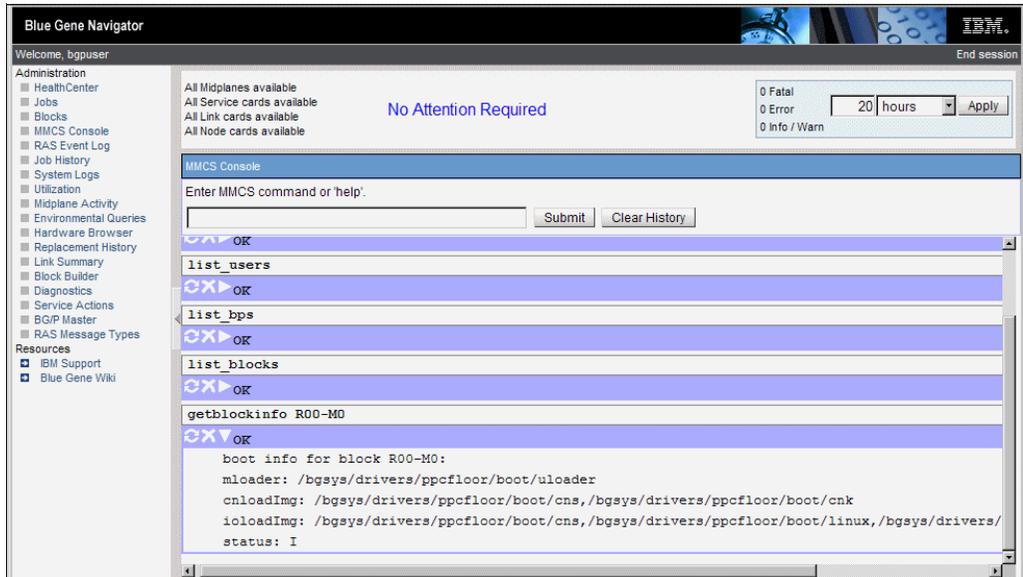


Figure 2-30 Navigator's MMCS interface

For more specific information about the MMCS server and MMCS console, see Chapter 5, “Midplane Management Control System” on page 91.

2.3.5 RAS Event Log

The RAS Event Log page offers various ways to view system events that generated RAS messages. In addition to filtering the messages, you can also choose how you view the messages. Figure 2-31 shows the most recent 50 messages in the event log in row format, which is the default view.

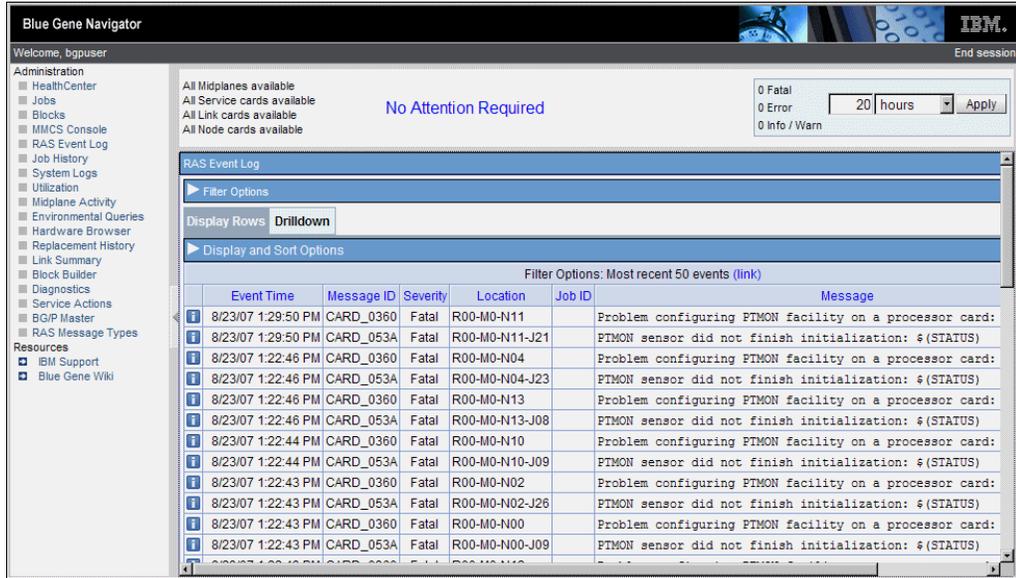


Figure 2-31 RAS Event Log

Between the *Filter Options* bar and the *Display and Sort Options* bar, you can select the method that you want to display the messages by clicking the Display Rows or Drilldown tab. Figure 2-32 shows the drill-down method. You can click the plus (+) symbol to drill down into the hardware to find the location of the error or errors.

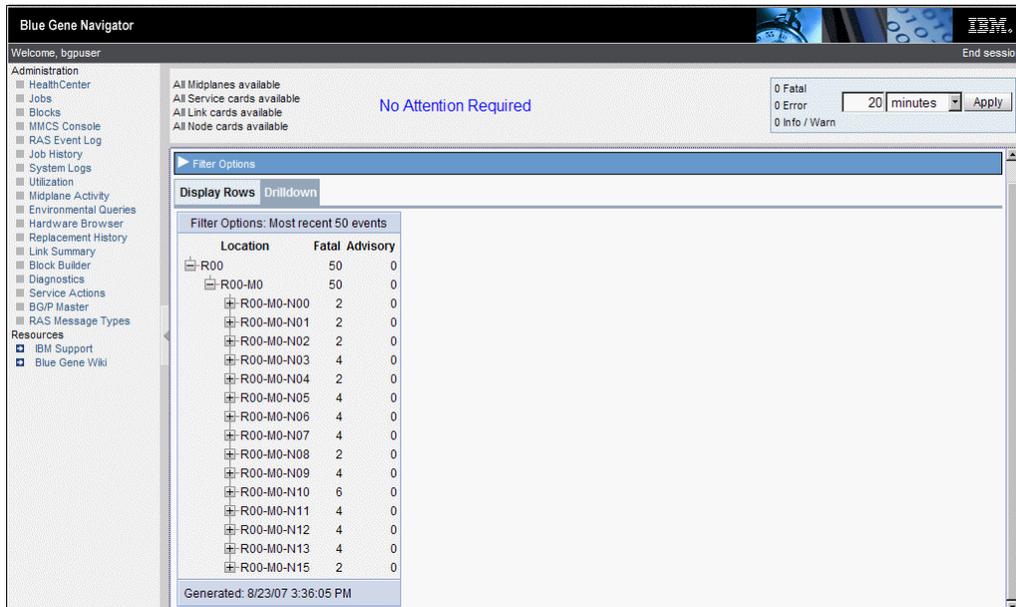


Figure 2-32 RAS Event Log Drilldown

Clicking the Filter Options twisty opens the page shown in Figure 2-33.

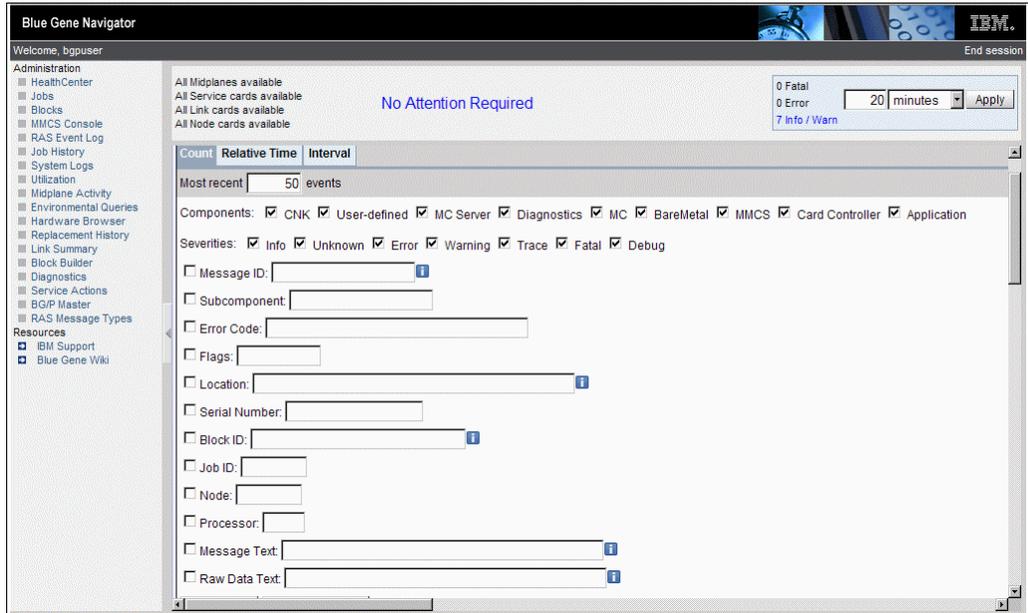


Figure 2-33 RAS Event Log Filter Options

Just below the Filter Options are the *Display and Sort Options*. Expanding this section gives you the page shown in Figure 2-34. By selecting or deselecting the various column heads you can customize the display. The drop-down menu allows you to select the column head that you sort by. Selecting either radio button sorts the results in ascending or descending order.

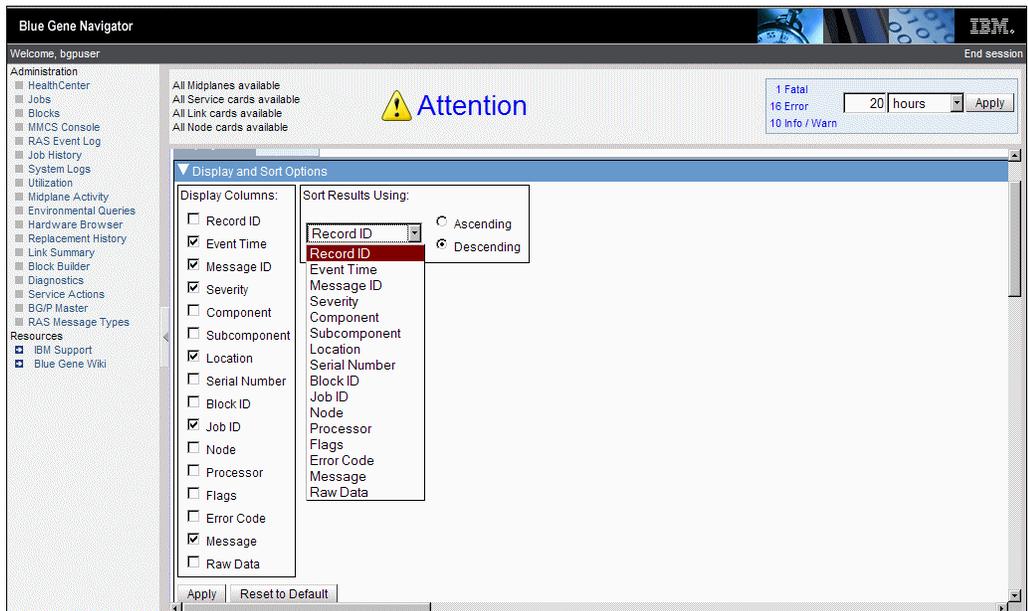


Figure 2-34 RAS event sort options

Notice the additional information icon in the first column (extreme left side) of the content area table in Figure 2-31 on page 41. Clicking that icon opens an RAS event detail page similar to the one shown in Figure 2-35. The details include the date and time the error occurred, a record ID, message ID and the severity of the event. Beyond the general information, the page contains information that relates to the specific event, including a description of the error and information about how to further handle the problem.

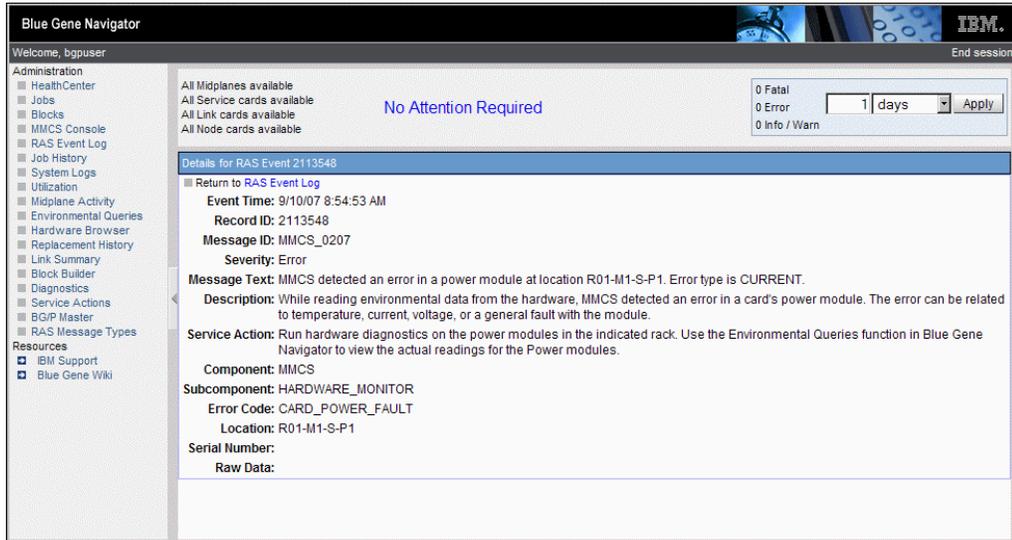


Figure 2-35 RAS event detail page

2.3.6 Job History

The Job History page, shown in Figure 2-36, by default shows the most recent fifty jobs that have run on the system. You can filter the data shown on the page by job type, specific user, block, or midplane. You can also query on the exit status, executable, or a single job identification number. There is also an additional filter, *HTC location*, that allows you enter up to a 32 character string to narrow your search results to a specific compute node or core.

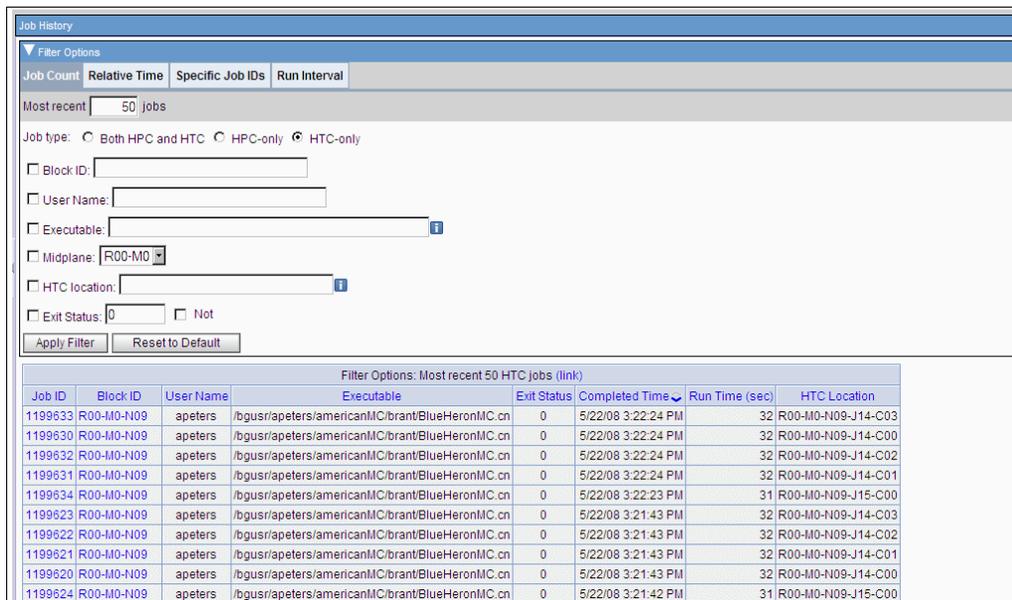


Figure 2-36 Job History

2.3.7 System Logs

Clicking the System Logs link in the navigation pane opens the page shown in Figure 2-37. From this page, you can view all the logs on the system—the historical logs (old) or the logs that are presently being written to (current).

By default, system logs are created in the `/bgsys/logs/BGP` directory. To view logs in other directories supply the path in the *Log directory* Filter Options box and click **Apply Filter**. This displays the contents of the selected directory that have a `.log` file extension. You can also filter the logs that are displayed by selecting a date and time stamp and then clicking **Apply Filter**.

The screenshot shows the Blue Gene Navigator interface. The top navigation pane includes links for Administration, HealthCenter, Jobs, Blocks, MMCS Console, RAS Event Log, Job History, System Logs, Utilization, Midplane Activity, Environmental Queries, Hardware Browser, Replacement History, Link Summary, Block Builder, Diagnostics, Service Actions, BGP Master, and RAS Message Types. The main content area displays the System Logs page with the following details:

- Status: No Attention Required
- Alerts: 0 Fatal, 8 Error, 0 Info / Warn
- Filter Options: Log directory: `/bgsys/logs/BGP`
- Containing timestamp: `9/10/07 9:40:13 AM`
- Buttons: Apply Filter, Reset to Default
- Current | All | Old
- Filter Options: Current log files in `/bgsys/logs/BGP`

Log name	Last changed	Log size
bgpsftsn-mcserver-current.log	9/10/07 9:40:05 AM	11 MB
bgpsftsn-bgdb0-mmcs_db_server-current.log	9/10/07 9:40:05 AM	12 MB
bgpsftsn-bgdb0-mpirund-current.log	9/10/07 9:38:29 AM	16 MB
bgpsftsn-bgdb0-navigator-current.log	9/10/07 9:09:01 AM	21 KB
bgpsftsn-bgpmaster-current.log	9/8/07 10:46:12 AM	2 KB
bgpsftsn-bgdb0-configureRealltime-current.log	9/7/07 11:19:23 AM	17 KB

Figure 2-37 System Logs

Each of the file names in the Log name column of the System Logs page is a link to the file itself. Clicking one of the filenames opens a window similar to the one shown in Figure 2-38. The Log Viewer feature of the Navigator allows you to view both current and historical logs and to perform a variety of searches within the log.

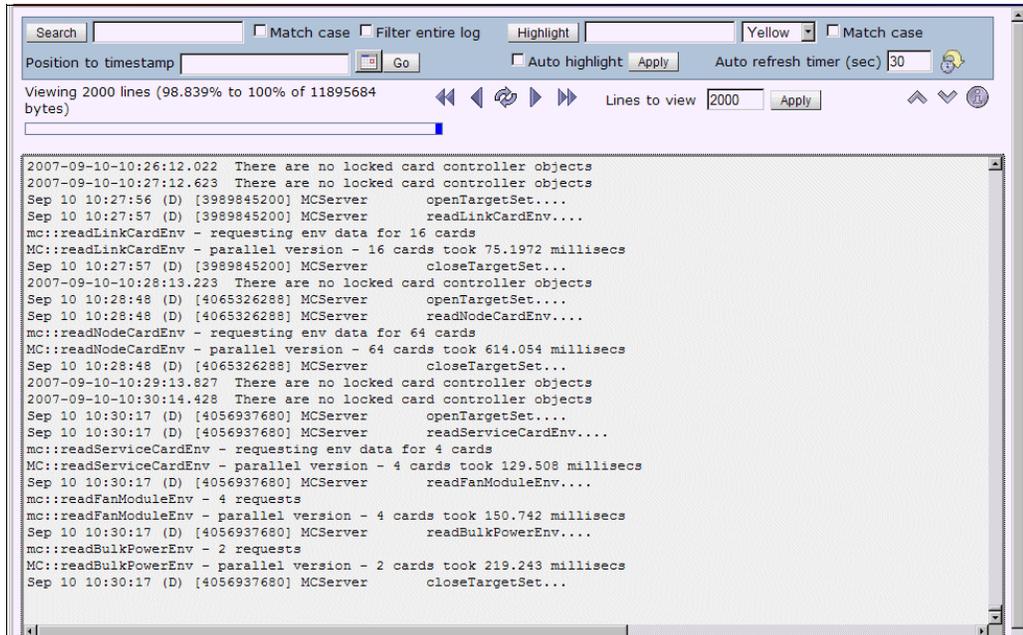


Figure 2-38 Log viewer window

Figure 2-39 shows a number of log viewer features. These features assist you in the following areas:

- ▶ **Movement:** You can move around in the log file and visualize where you are in the log file. Movement can be by paging backwards and forwards or by moving to a specific time stamp or other search string.
- ▶ **Refresh:** When viewing a current (live) log file, it might be growing in size or rolling over to become a new log file, while you are viewing it. You can refresh your log file view manually or automatically.
- ▶ **Filtering:** You can select only the log file lines of interest, in a manner similar to the Linux **grep** command. This filtering can be applied to the entire log file or to the lines that are currently being viewed.
- ▶ **Highlighting:** Two independent highlighting schemes are available. In Figure 2-39, the auto highlight feature displays lines in various colors (for example, error messages are displayed in red), and the manual highlight feature is highlighting the text *job* in line.
- ▶ **Hints:** Hover your mouse over any of the log file viewer icons, and the viewer provides a *hint* that describes that icon's function. In addition, if you hover your mouse over the information (*i*) icon, the log viewer displays the message code descriptions as shown in Figure 2-39.

You can copy and paste from the log view to any of the form fields. For example, you can search for a specific message, copy that message's times stamp text into the *position to timestamp* box, and then click **Go** to display the full log beginning at that time stamp.

The search box provides two different modes of searching, depending on whether the *filter entire log* box is selected. If the box is *not* selected, clicking the Search button causes your browser to search the lines that are currently being viewed. This is identical to using your browser Find function, except that the log viewer limits the search to the log file lines.

The log viewer does an entirely different type of search when the *Filter entire log* box is selected. In this case, the log viewer does a server-side search, returning only those lines that

match the search string. All search strings are fixed strings. The log viewer does not recognize wildcard characters or regular expressions.

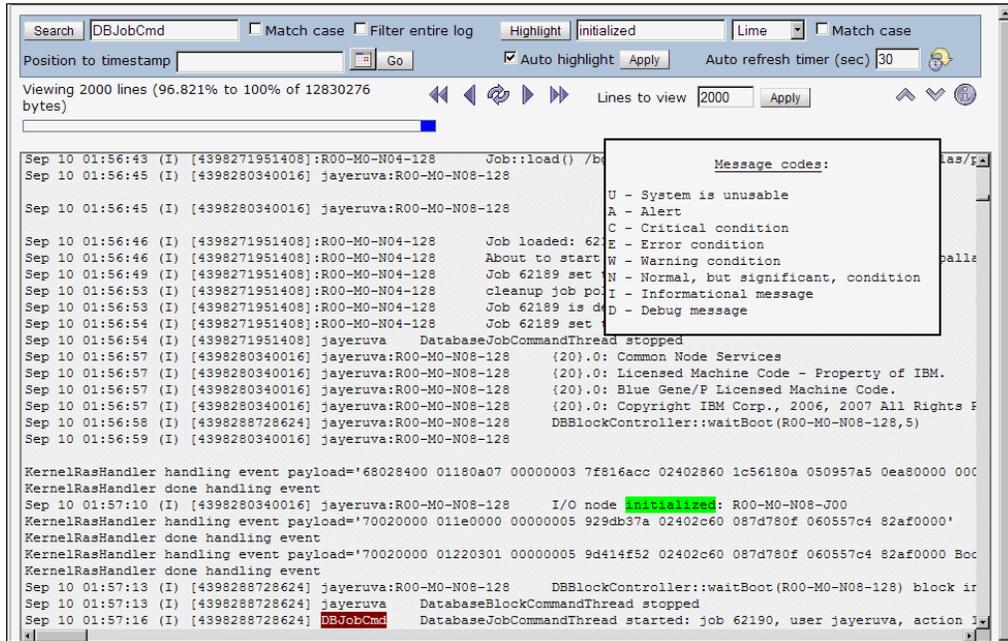


Figure 2-39 Log viewer features

2.3.8 Utilization

Figure 2-40 shows the page that displays when you click **Utilization**. This page reports the system utilization, by midplane, for the time specified in the filter options. The Filter Options on this page allow you to select the amount of time that is reflected in the report. The options include *Relative Time* and *Interval*. Relative Time allows you to select a period of time, beginning at the present time, that you want to go back in history. You supply the numeric value, and the drop-down menu gives you a choice of seconds, minutes, hours, and days.

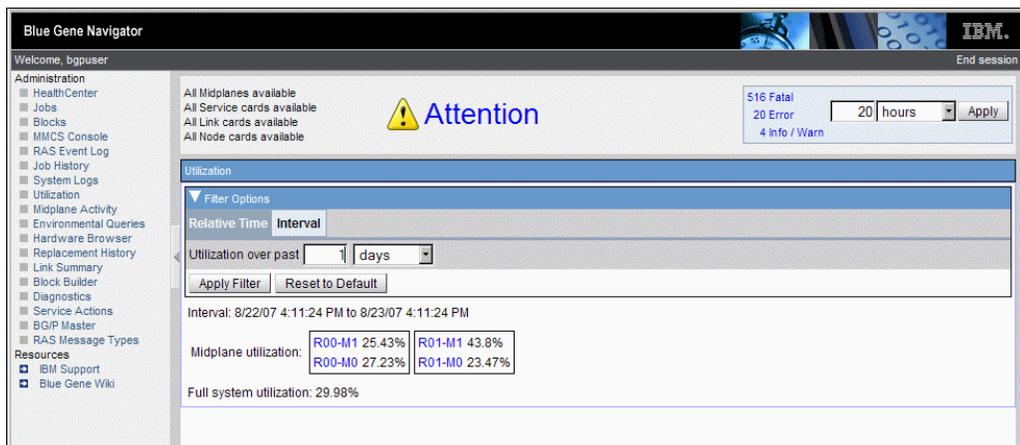


Figure 2-40 Utilization page

The Interval option allows you to pick a specific period of time on which to base the results. You can enter the dates manually using the format dd/mm/yy h:mm:ss AM/PM. The other method is to click the calendar icon and select the date from the calendar drop-down that displays. The default time period for both options is one day.

Regardless of the option that you choose, the results are returned by midplane. The location of the midplane is followed by the percentage of utilization for the time period you chose. The midplane location is actually a link. Clicking the location link queries the database and returns a job history page that contains all of the jobs that ran during the specified time period (Figure 2-41).

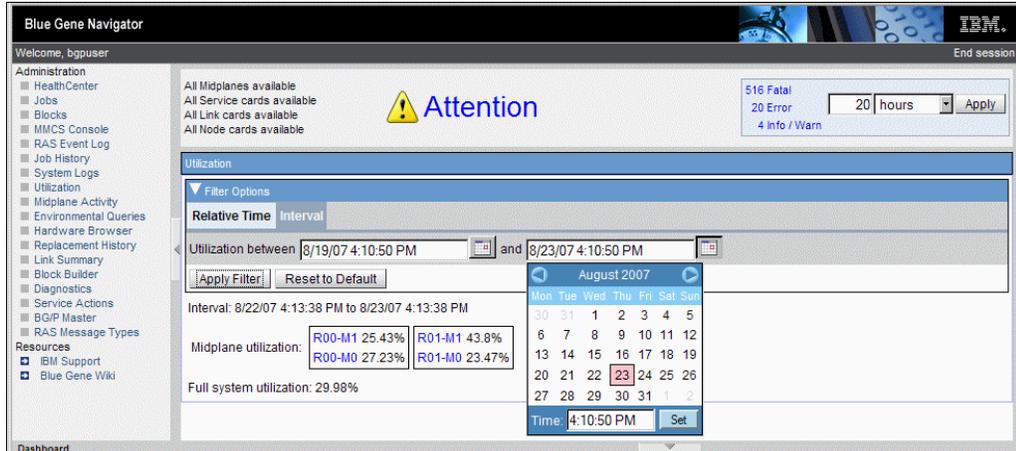


Figure 2-41 Interval option

2.3.9 Midplane Activity

The Midplane Activity page shows all the current activity on the system. Clicking the link opens the page shown in Figure 2-42. You can filter your output to include the entire system (all midplanes), a single midplane, a selected block or an HTC pool. The tool reports any blocks in use on the hardware, as well as the status and the current user of that block. You can also determine whether the midplane's link cards are being used to pass through to other midplanes to complete a torus. Any jobs currently running on the midplane also display. Both the *Block* and *Job* columns contain links to their respective summary pages.

The screenshot shows the 'Midplane Activity' page. It features a 'Filter Options' section with checkboxes for 'Midplanes including', 'Used by block', and 'HTC Pool:'. Below the filters, a table displays the activity for various midplanes. The table has columns for Midplane, Block, Block Status, Nodes, Passthrough, User, Job Info, Job Status, and HTC Pool. The data is as follows:

Midplane	Block	Block Status	Nodes	Passthrough	User	Job Info	Job Status	HTC Pool
R00-M0	R00-M0-N14	Initialized	32		megeri	2 jobs	HTC - SMP	R00-M0-N14
R00-M1	R00-M1	Initialized	512		bknuds	1 job	HTC - SMP	POOL1
R01-M0	R01	Initialized	512		bknuds	2,047 jobs	HTC - VN	POOL1
R01-M1	R01	Initialized	512		bknuds	2,048 jobs	HTC - VN	POOL1
R10-M0	R10	Initialized	512		megeri	4288	Running	
R10-M1	R10	Initialized	512		megeri	4288	Running	
R11-M0								
R11-M1								

Figure 2-42 Midplane Activity

Figure 2-42 shows that the Filter Options for Midplane activity include a filter for HTC Pools. Looking further down the page, block R01 which spans midplanes R01-M0 and R01-M1, has been booted in virtual node mode for HTC in pool POOL1. 4,095 HTC jobs are running on this rack, with 2,047 of them on R01-M0 and 2,048 on R01-M1. It also shows that the pool POOL1 includes another midplane booted in SMP mode. Clicking on the job count for the midplane will show the Job Summary page with the filter set to display those jobs.

2.3.10 Environmental Queries

The Environmental Queries section of the Navigator provides access to historical data collected by the hardware monitoring functions in the MMCS server. Figure 2-43 shows the main page and filter options that are available when you click the Environmental Options link in the navigation pane. Clicking any of the tabs opens a new page that shows the most current readings taken by the Hardware Monitor. Expanding the Filter Options on each of the pages shows the options that are available relative to the hardware tab that is selected. For more information about the Environmental Monitor, see 5.7, “Environmental Monitor” on page 104.

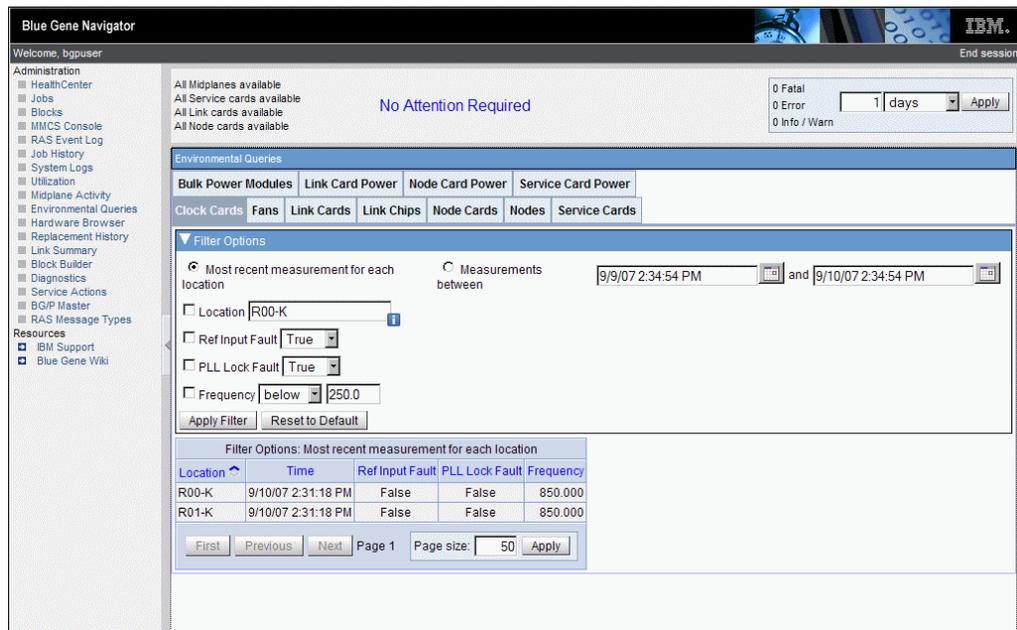


Figure 2-43 Environmental Queries

2.3.11 Hardware Browser

The initial Hardware Browser page contains information about the system's configuration (Figure 2-44). This information is a combination of several of the DB2 tables:

- ▶ BGP MACHINE
- ▶ BGPETHGATEWAY
- ▶ BGP MACHINESUBNET

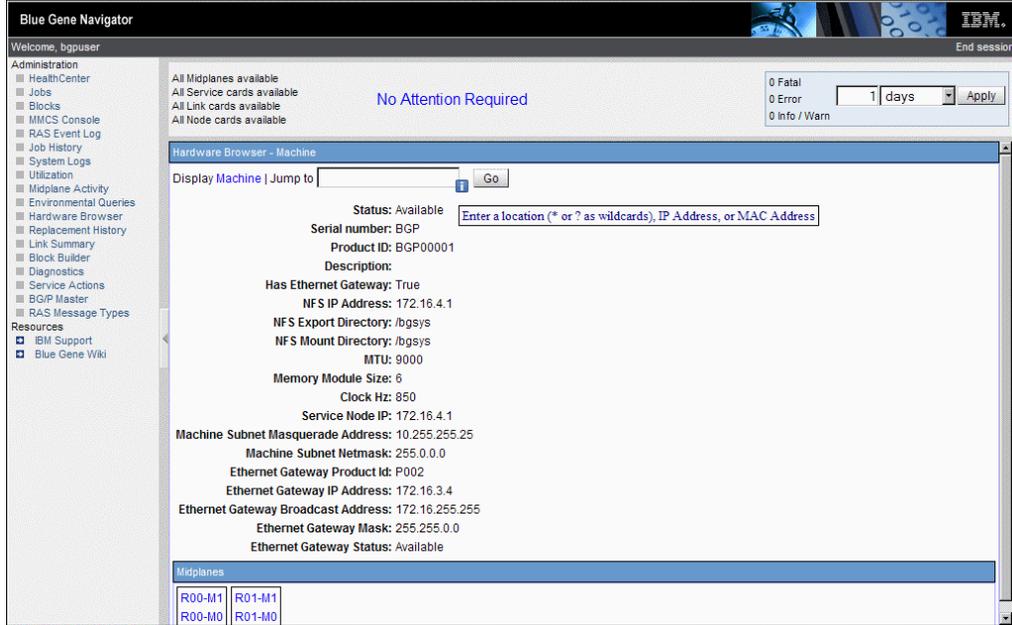


Figure 2-44 Hardware Browser link

Click any of the links at the bottom of the page and information pertaining to the specified rack displays (Figure 2-45). Any hardware that is not available is flagged with the appropriate status. This page contains information about each of the midplanes, Bulk Power Modules (BPMs), Clock cards, and clock signal distribution.

The screenshot shows the Blue Gene Navigator interface. At the top, it says 'Welcome, bgpuser' and 'End session'. The left sidebar contains navigation options like 'Administration', 'HealthCenter', 'Jobs', 'Blocks', etc. The main content area shows 'All Midplanes available', 'All Service cards available', 'All Link cards available', and 'All Node cards available'. A status bar indicates 'No Attention Required' with 0 Fatal, 7 Error, and 0 Info / Warn items. Below this, there are tables for 'Midplanes', 'Bulk Power Supply', 'Clock Card', and 'Clock Cable Connections'. The 'Midplanes' table lists R01-M0 and R01-M1 as available. The 'Bulk Power Supply' table lists R01-B-P0 through R01-B-P8, with R01-B-P2 marked as 'Error'. The 'Clock Card' table lists R01-K as available. The 'Clock Cable Connections' section notes 'This rack is the master clock source.' and 'Clock Propagated to Rack: R00'.

Figure 2-45 Rack information

Click one of the midplane links to display all of the cards in that midplane, see Figure 2-46. This page contains information about the Node cards, Service card, and Link cards.

The screenshot shows the Blue Gene Navigator interface with a warning icon and 'Attention' text. The left sidebar is the same as in Figure 2-45. The main content area shows 'All Midplanes available', 'All Service cards available', 'All Link cards available', and 'All Node cards available'. The status bar shows 6 Fatal, 0 Error, and 0 Info / Warn items. Below this, there are tables for 'Node Cards', 'Service Card', and 'Link Cards'. The 'Node Cards' table lists R01-M0-N00 through R01-M0-N15, all available. The 'Service Card' table lists R01-M0-S as available. The 'Link Cards' table lists R01-M0-L0 and R01-M0-L1 as available.

Figure 2-46 Hardware Browser midplane level

Clicking the available links drills down further on each of the cards. For example, if you click one of the node cards, you open a page as shown in Figure 2-47 that contains each of the compute card locations that particular node card. This page contains information about the amount of memory on each of the nodes, voltages, identification number, and whether the card in this location has ever been replaced.

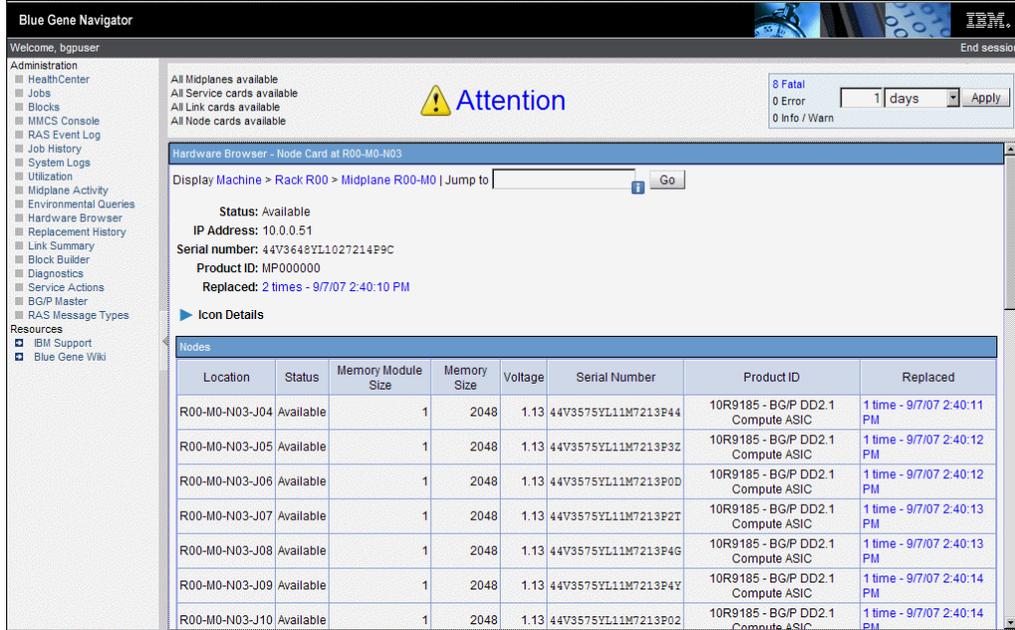


Figure 2-47 Hardware Browser node card

Click the icon details twisty to open the section shown in Figure 2-48. This information pertains to the Icon and Palomino chips located on the card.



Figure 2-48 Icon Details

2.3.12 Link Summary

Clicking the Link Summary link in the Navigation pane shows a diagram of the links created with data cables in the X, Y, and Z dimensions. Figure 2-49 shows the links in a four rack configuration. These diagrams are especially useful when creating blocks because you can see the progression of the cables through the rows and columns.

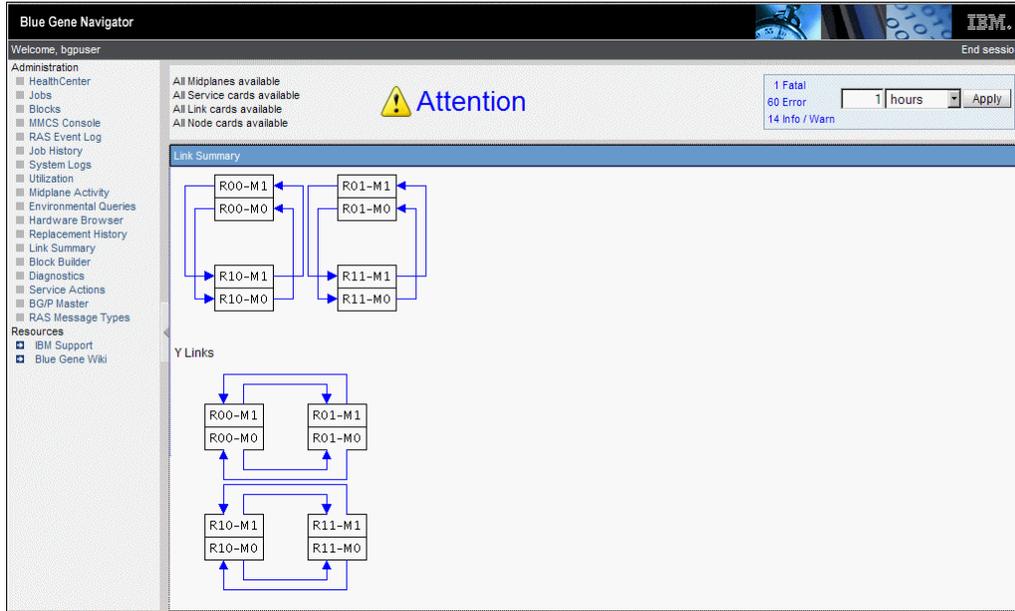


Figure 2-49 X and Y Links

Figure 2-50 shows how the Y and Z dimensions are cabled in a two-rack system.

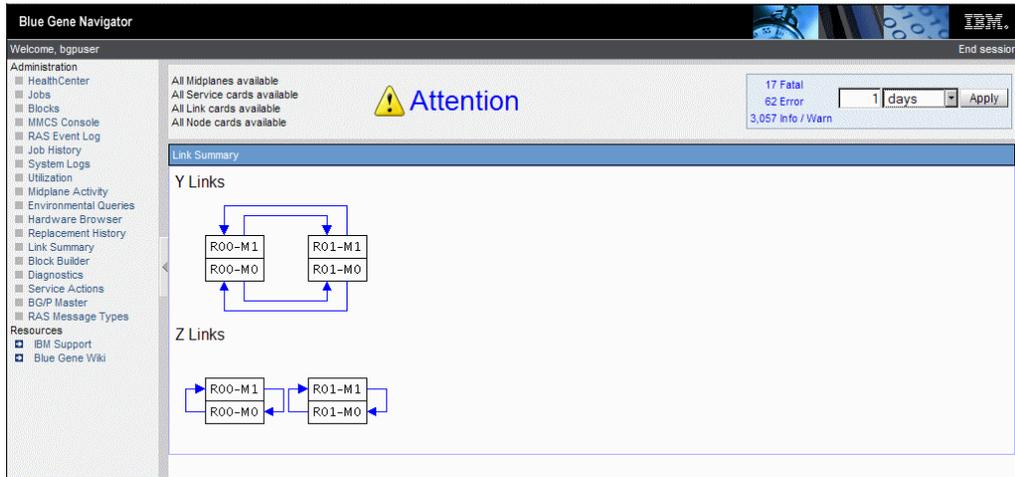


Figure 2-50 Y and Z Links

2.3.13 Block Builder

Figure 2-51 shows the Block Builder page. This page provides a graphical interface to create any block needed on a Blue Gene/P system. For instructions on creating blocks using this tool, see 4.2, “Blue Gene Navigator” on page 81.

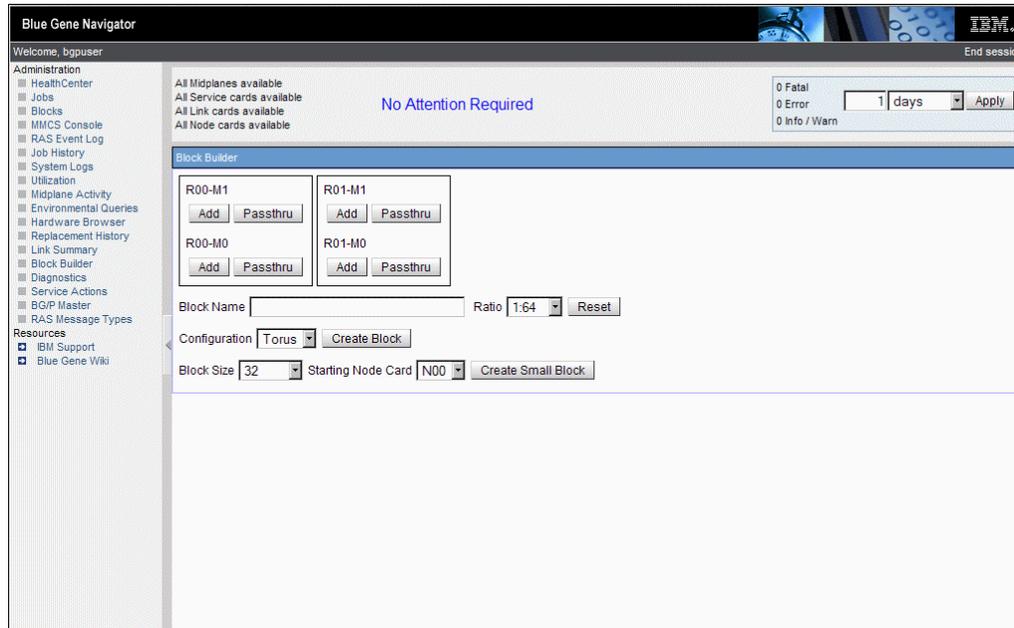


Figure 2-51 Block Builder

2.3.14 Diagnostics

The Diagnostics page, shown in Figure 2-52, provides a method to create new Diagnostic runs and also view runs that are currently running or have completed. The Summary tab shows you any Diagnostic runs that are active and also the status of the most recent completed Midplane, Link, and User-defined Block run. You can also initiate a new run from this tab. The Locations and Completed Runs tabs allow you to view previous runs. The Filter Options on the Locations tab allows you to narrow the results based on location, hardware status, time of the run or whether hardware was replaced. On the Completed Runs tab, you can filter on the status of the runs, time, midplane, racks, or specific blocks.

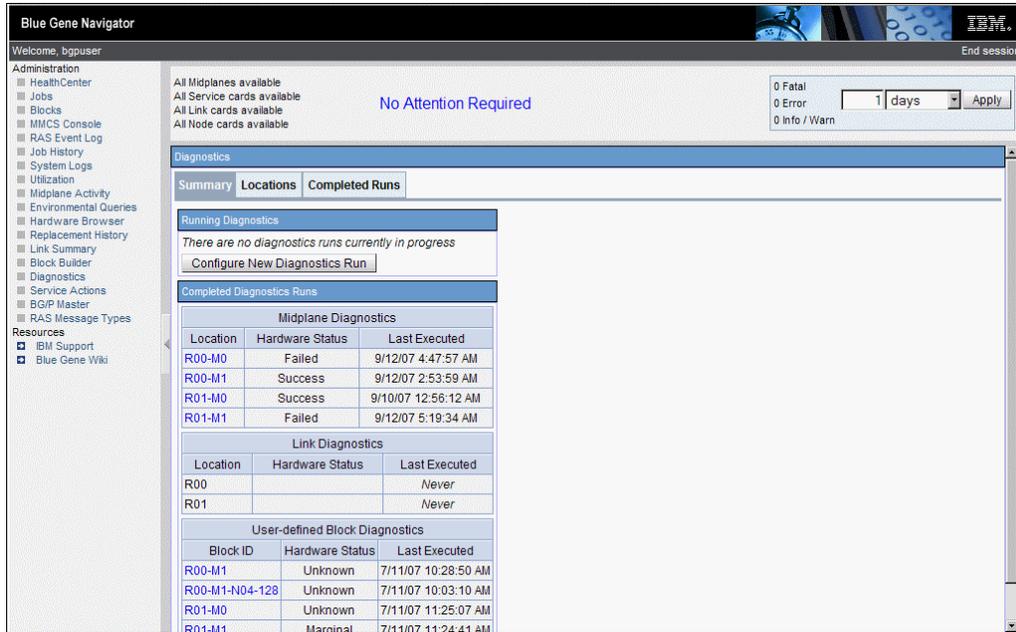


Figure 2-52 Diagnostics home page

2.3.15 Service Actions

Navigator allows administrators to initiate Service Actions from the browser. You can also query the database through the Navigator to view current or historical Service Actions. The Filter Options also allow you to limit query results by time stamp, user, or location. Figure 2-53 shows the Service Action page with the Filter Options expanded. More information about Service Actions is available in Chapter 9, “Service Actions” on page 139.

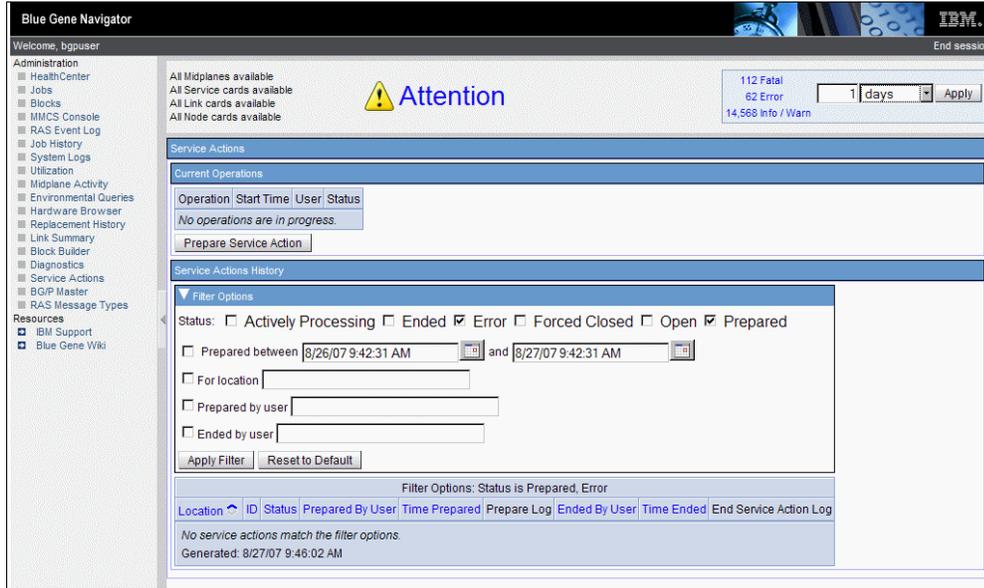


Figure 2-53 Blue Gene/P Service Actions

2.3.16 BG/P Master

Navigator’s interface to the BGPMaster processes is shown in Figure 2-54. You can start each of the servers that are monitored by BGPMaster, except navigator_server, from this page.

BG/P Master						
Name	Status	PID	Action	Auto Start	Start Command	Status Message
mcserver	Started	8097	Stop Restart	Yes	startmcserver	
mmcs_server	Started	8101	Stop Restart	Yes	startmmcs	
mpirund	Started	8107	Stop Restart	Yes	startmpirund	
navigator_server	Started	8111	Stop	Yes	startnavigator	
realtime_server	Stopped		Start		startrealtime	need Administrator to restart. Down for 31962 seconds
submit_server	Started	8127	Stop Restart	Yes	startsubmitserver	

Figure 2-54 Blue Gene/P Master

You can stop all of the servers. However, keep in mind that if you stop the navigator_server, you no longer have a connection to restart the processes. In this case, you have to restart, at the very least, the navigator_server from the command line.

2.3.17 RAS Message Types

Figure 2-55 shows the RAS Message Type page. This page displays RAS messages that have occurred on your system during a specified time period. The default shows all messages for the last 24-hour period. You can sort the results by clicking any of the column heads.



Figure 2-55 RAS Message Types

Figure 2-56 displays the possible Filter Options available in the RAS Message Types list.

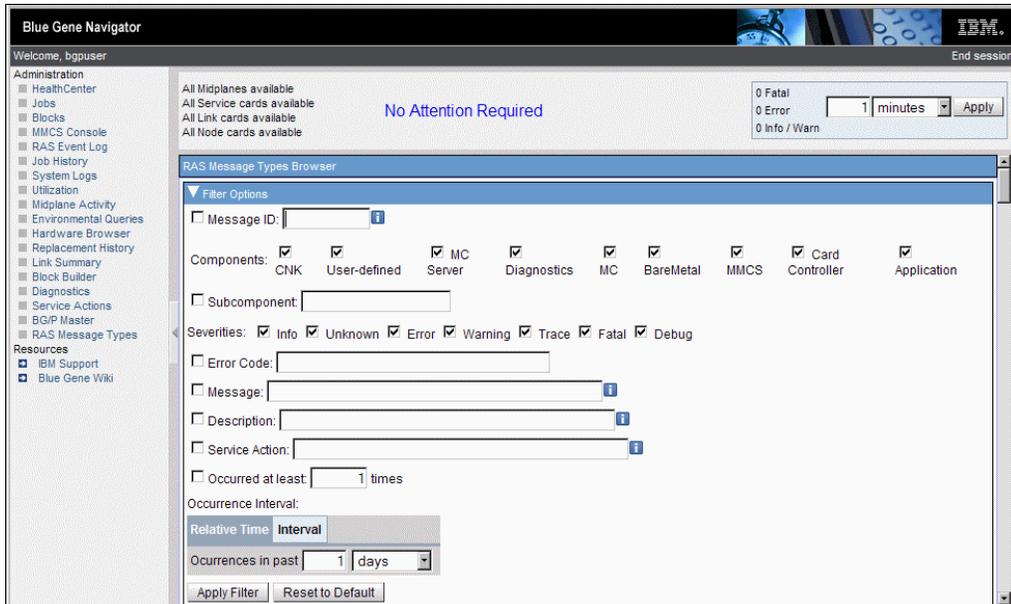


Figure 2-56 RAS Message Types Filter Options

The first option allows you to filter by *Message ID*. The field accepts up to 10 characters, and you can also use wildcards (* or ?). You can search the messages by *Components* or message *Severities* by selecting the source or level of message that you want to see. By default, all of the options are selected, and you can eliminate any of the options by deselecting them.

If you look back at Figure 2-55, you see the *Error Code* column in the center of the page. The error code strings are also searchable by checking the box next to Error Code and then supplying the string in the box provided. Similarly, you can use each of the next three options (*Message*, *Description* and *Service Action*) as a search argument by checking the appropriate box and providing a string. Each of the fields allow you to enter up to 1024 characters or wildcards.

The next option, *Occurred at least: x times*, provides a method of searching based on the number of occurrences a message has appeared. Selecting the box and providing a threshold value returns the number of times a message has been entered in the RAS database in the time period specified. Using this filter option gives you the opportunity to evaluate whether errors are random or if there might be a trend developing.

The last option is to set the time period to be searched. You can search back in time starting from the current time, which is the Relative Time tab. The other method is to click the Interval tab, which gives you the opportunity to supply a start and stop time to narrow your search. Your search can be broad by selecting only a component or a message severity, or it can be very defined by selecting more than one option and providing specific messages and time periods.

2.4 Customizing Blue Gene Navigator

As we mentioned in the previous sections, there are several areas of the Blue Gene Navigator interface that are customizable. In this section, we describe how to add to or change the interface to suit your environment.

When you install the system, the default location of the configuration file is `/bgsys/drivers/ppcfloor/navigator/navigator-config.xml`. Depending on your policy, you might want to create a local configuration file (`navigator-config.xml`), which is usually located in `/bgsys/local/etc`, with your additional configurations. Modifications made in this file will not be affected by driver upgrades.

2.4.1 Adding resource links

You can replace existing or define additional resource links to be displayed in the navigation pane of the Navigator. Example 2-14 demonstrates how to add a new link to the Navigation pane. You must supply the following values in the resource-link container in your local configuration file.

Name	Must be unique for each definition. When overriding values for an existing definition the same name must be given.
Enabled	A Boolean value that indicates whether the link displays. If true, then the link displays in the browser. The default is true.
Label	This displays as the text of the resource link in the Navigation pane.
URL	The URL that to which the browser is directed when the link is selected.
New Page Indicator	If set to true, a new page opens when the user clicks the link. The default is true.
Navigation Image	Overrides the default bullet image. Should be a URL that the browser can use to fetch an 8x8 pixels image. Displays next to the link in the Navigation area.
Style	Overrides the default style. Should be Cascading Style Sheets (CSS).

Priority Resource links displays in order of increasing priority, from first to last. This must be an integer.

Example 2-14 Resource-link container

```
<resource-link>
  <name>New Link</name>
  <enabled>>true</enabled>#not required unless setting to 'false'
  <label>My New Link</label>
  <url>http://www.NewLink.com</url>
  <new-page>true</new-page>#not required unless setting to false
  <navigation-image>http://example.com/image1.png</navigation-image>
  <style>color: green;</style>
  <priority>2500</priority>
</resource-link>
```

There is one default resource link that is defined in the Navigator configuration file that provides a link to the IBM Blue Gene support home page. Example 2-15 shows the configuration file contents for this link.

Example 2-15 Default resource link

```
<resource-link>
  <name>IBMSupport</name>
  <label>IBM Support</label>
  <url>http://www.ibm.com/servers/eserver/support/bluegene/index.html</url>
  <priority>1000</priority>
</resource-link>
```

As an example of overriding a value of the default resource link, this link could be disabled by setting the enabled tag to false in the local configuration file (Example 2-16).

Example 2-16 Disabling a resource link

```
<resource-link>
  <name>IBMSupport</name>
  <enabled>>false</enabled>
</resource-link>
```

2.4.2 Chart Plug-ins

You can use two types of plug-ins to customize the Navigator interface for your environment: Attention Plug-ins and Chart Plug-ins. You can use *Attention Plug-ins* to cause the Attention indicator in the Attention area to display and to have messages display in the Health Center. You can use *Chart Plug-ins* to add charts to the Dashboard.

There are three chart plug-ins shipped by default: submitted jobs, software RAS events, and system utilization (Example 2-17).

Example 2-17 Default plug-ins

```
<chart-plugin>
  <name>jobs</name>
  <class>com.ibm.bluegene.navigator.charts.JobsChartGenerator</class>
  <priority>1000</priority>
</chart-plugin>
<chart-plugin>
```

```

    <name>ras</name>
    <class>com.ibm.bluegene.navigator.charts.RasChartGenerator</class>
    <priority>2000</priority>
  </chart-plugin>
</chart-plugin>
<chart-plugin>
  <name>utilization</name>
  <class>com.ibm.bluegene.navigator.charts.UtilizationChartGenerator</class>
  <priority>3000</priority>
</chart-plugin>

```

Creating a plug-in

Whether creating an attention plug-in or a chart plug-in, you need to override a class that is provided in `navigator.jar`, which is shipped in `/bgsys/drivers/ppcfloor/navigator/catalina_base/webapps/BlueGeneNavigator/WEB-INF/lib`. Build your classes with `navigator.jar` in the classpath. Then create a jar file containing the classes and a definition file.

Attention plug-in

Attention plug-ins extend `com.ibm.bluegene.navigator.attention.AbstractAttentionPlugin`. When Navigator starts, it creates an instance of your class using the default constructor and then calls `initialize()` with an `com.ibm.bluegene.navigator.config.AttentionPluginConfig` object. From the `AttentionPluginConfig` passed to `initialize()`, you can get the properties that the user has configured for your plug-in.

Whenever a page is requested, the Navigator can call the `isAttentionRequired` method (it will not call it if the attention state is already known). This must return true to indicate that attention is required, otherwise false.

When the Health Center page is requested, the Navigator calls the `getAttentionMessages` method that should return a list of `AbstractAttentionPlugin.MessageInfo` objects. The `MessageInfos` returned displays in the Health Center. The returned messages are labeled with the string returned by `getLabel`.

The attention plug-in's definition file must have an `attention-plugin` element with a unique name and reference your implementation class. Example 2-18 gives an example attention plugin configuration file.

Example 2-18 Attention plugin file

```

<?xml version="1.0" standalone="yes">

<navigator>
  <attention-plugin>
    <name>Sample1</name>
    <class>com.example.plugin</class>
  </attention-plugin>
</navigator>

```

Chart plug-in

Chart plug-ins extend `com.ibm.bluegene.navigator.charts.AbstractChartGenerator`.

When Navigator starts, it creates an instance of your class using the default constructor and then calls `initialize()` with a `com.ibm.bluegene.navigator.config.ChartPluginConfig` object. From the `AttentionPluginConfig` passed to `initialize()`, you can get the properties that the user has configured for your plugin. The methods that you want to override are `getLabel` and

`getChartInfo`. `getLabel` supplies Navigator with the text to display above the chart in the Dashboard. `getChartInfo` can be called every time a request is made to Navigator (it will not be called if the Dashboard is hidden) and supplies Navigator with the information that it sends to the client so that the client will request the image. You might also want to override `getLegendHtml` to provide a legend to be displayed next to the chart.

For most types of charts, it will probably not be fast enough to generate the image on every request. For this reason, a class is supplied that generates the image in a background thread every once a while and just sends back the cached result on each request. This class is `com.ibm.bluegene.navigator.config.AbstractTimeoutThreadChartGenerator` which extends `AbstractChartGenerator`.

To use this class, you override several abstract methods:

<code>initializeInstance()</code>	Called when Navigator starts up. You can get any property values from the supplied <code>ChartPluginConfig</code> .
<code>generateChartInfo()</code>	Called from a thread when the timeout occurs to generate the new chart info.
<code>applyState()</code>	Called with the result of <code>generateChartInfo()</code> .
<code>getReqChart()</code>	Called whenever a page is requested to return the cached <code>ChartInfo</code> . You will probably get the cached <code>ChartInfo</code> from what was set by <code>applyState()</code> .
<code>getThreadName()</code>	Used when debugging to make it easy to find the thread that is calling <code>generateChartInfo</code> and <code>applyState</code> .

You will still want to override `getLabel` and might want to override `getLegendHtml`.

Installing Navigator plug-ins

There are two parts to the plugin: a definition file and a jar file. The definition file is named `navigator-config.<id>.xml`, where `<id>` is a unique name. Put this file in `<configpath>` (`/bgsys/local/etc/` by default). The jar file is put into `<navigatorlibs>` (`/bgsys/local/lib/navigator` by default). Make sure that Navigator can read these files.

Configuring Navigator plug-ins

Take a look at the plugin's configuration file in `<configpath>` (`/bgsys/local/etc/` by default). If you want to override any options, you will need to know the type and name of the plugin, which you can figure out from its configuration file. The type is indicated by whether the configuration file contains an `attention-plugin` or `chart-plugin` element. The name is in the name element under that element. To override an option, add the same `attention-plugin` or `chart-plugin` element to your local Navigator configuration file `<configpath>/navigator-config.xml` (`/bgsys/local/etc/navigator-config.xml` by default) with the new value for the option. The options for both types of plug-ins are:

<code>enabled</code>	Set to false to disable. If disabled the plugin will not be loaded. Defaults to true.
<code>class</code>	The fully-qualified name of the class that implements the plugin.

Plug-ins can also define properties to define other options. Properties are defined in property elements (Example 2-19).

Example 2-19 Property elements

```
<chart-plugin>
  <property>
    <name>prop-name</name>
```

```

    <value>prop-value</value>
  </property>
</chart-plugin>

```

Additional options for chart plug-ins are:

priority	Defines the order that the chart will appear in the dashboard frame. 0, the default, will be at the left side of the frame.
timeout	The time between chart generations, or default to use the default.

As an example of overriding an option for a plugin, consider setting the priority to 15 for a chart plugin whose name is Sample1. The local Navigator configuration file would be similar to Example 2-20.

Example 2-20 Local configuration file

```

<navigator>
  <chart-plugin>
    <name>Sample1</name>
    <priority>15</priority>
  </chart-plugin>
</navigator>

```

More resources

The JavaDocs for classes used by plug-ins are shipped in `navigator/doc/plugin/javadoc`.

A sample attention plugin is shipped in `navigator/doc/plugin/sample/attention_plugin`. The `DiagsIntervalAttentionPlugin` will cause the Attention indicator to be displayed if diagnostics has not run successfully on a midplane in some interval (the default is 1 month). The interval can be overridden by setting the 'interval' property for the plugin. The value of the 'interval' property must be an SQL Labeled Duration (e.g., '1 MONTH').

A sample chart plugin is shipped in `navigator/doc/plugin/sample/chart_plugin`. The `NodeTempChartPlugin` charts the maximum temperature of any node for each day for the past 5 days. It uses JFreeChart to generate the image.

2.4.3 Problem determination

This section covers a few of the more common issues that arise with Navigator.

Navigator logging level

Navigator uses three loggers:

- ▶ `com.ibm.bluegene.navigator`
- ▶ `com.ibm.bluegene.navigator.db`
- ▶ `com.ibm.bluegene.navigator.mmcs`

The `com.ibm.bluegene.navigator` logger is used for general logging messages. The `com.ibm.bluegene.navigator.db` logger is used for database queries. Set the level for `com.ibm.bluegene.navigator.db` to FINE and the database statements used will be logged. The `com.ibm.bluegene.navigator.mmcs` logger is for MMCS Console log messages. Set the logging level for `com.ibm.bluegene.navigator.mmcs` to FINE and the commands sent to `mmcs_db_server` will be logged.

There are two ways to set the logging level: static and dynamic. Static log level setting requires that the Navigator be restarted, and the logging level will continue to be used across Navigator restarts. Dynamic log level settings are made while Navigator is running and take place instantly, but the setting is discarded when Navigator is restarted.

To use the static method, edit the file `/bgsys/drivers/ppcfloor/navigator/logging.properties` and set the logging levels for the different loggers. For example, to set the logging level for the `com.ibm.bluegene.navigator` logger, remove the `#` from in front of the line for `com.ibm.bluegene.navigator.level` and set the logging level after the `=`.

To use the dynamic method, go to change the last part of the URL to `about.jsp` and hit enter to load the page. Then expand the Logging Configuration section, and the logging level for each logger can be set using the drop-down, then select **Apply**.

Authentication problems

If users are unable to authenticate, then the problem is usually a) users are not in the right group, or b) the user that Tomcat is running under does not have the right authority. To check for (a), look in the local navigator configuration file (`/bgsys/local/etc/navigator-config.xml` by default) and make sure that the correct groups are listed. To check for (b), check near the top of the Navigator log file, where the groups that the user that started it is a member of are printed and make sure that the user or group has authority required to do PAM authentication. The most common PAM setup uses the `unix2` PAM module and so requires that the user be able to read `/etc/shadow`. (Note that if you change the groups for a user it will not be picked up by existing sessions, so sign off before restarting Navigator and check the Navigator log again).

First make sure that the user is in the right groups, otherwise they will be rejected. Check the group membership using `groups <username>`, then look in the local navigator configuration file (`/bgsys/local/etc/navigator-config.xml` by default) and make sure that one of the groups that they're in is listed in the group settings. Check `/var/log/messages` to make sure that PAM isn't printing out something.

JAAS tracing

If the group membership is correct, the next step is to check that users are not being rejected during the JAAS step. To get the trace messages for the JAAS module, edit the file `/bgsys/drivers/ppcfloor/navigator/catalina_base/conf/login.config` and add `debug=true` before the `;` (Example 2-21). The debug text is printed to the regular Navigator log.

Example 2-21 Setting debug mode

```
BlueGeneNavigator {
    com.ibm.bluegene.auth.module.LinuxPAMLoginModule required
    config="/bgsys/local/etc/navigator-config.xml,/bgs/dist/etc/navigator-config.xml"
    library="/bgsys/drivers/ppcfloor/navigator/bluegenenavigator.so" debug=true;
};
```



BGPMaster

BGPMaster is the basic process that monitors the Blue Gene/P's control processes. There are several jobs that run under the BGPMaster process including:

- ▶ mcServer
- ▶ Midplane Management Control System (MMCS)
- ▶ mpirund
- ▶ Blue Gene Navigator
- ▶ Real-time Server

In this chapter, we explain the function of the BGPMaster and how to start and stop the BGPMaster process and the jobs that run under it. We also examine the logs to which each of the jobs writes.

3.1 BGPMaster

BGPMaster is a daemon running on the Service Node that monitors and restarts any failed control system components. By design the components that BGPMaster can control are mcServer, mmcs_db_server, mpirund, navigator_server, and the realtime_server. To ensure proper authority to log files, BGPMaster should be started by a member of the bgpadmin group.

3.1.1 BGPMaster configuration file

When the BGPMaster startup script is initialized, it reads from the bgpmaster.init file. By default, the script looks in the /bgsys/local/etc directory for any local configurations. You need to make changes to the default characteristics in the .init file in that directory so that release upgrades will not affect expected behavior in your environment.

Example 3-1 shows the contents of a bgpmaster.init file. Commenting out the start line (inserting a # character) causes the server on that line to be skipped in the startup process.

Example 3-1 The bgpmaster.init file

```
# This bgpmaster.init file has lines of the format
#
#   # comment
#   define server-name start-command
#   start server-name
#
# The servers are initially started in the order that they are defined
#
define    mcserver          startmcserver
define    mmcs_server      startmmcs
define    mpirund          startmpirund
define    navigator_server  startnavigator
define    realtime_server   startrealtime

start    mcserver
start    mmcs_server
start    mpirund
start    navigator_server
# start  realtime_server
# Uncomment start realtime if you or your scheduler are using the
# real-time bridge APIs and have configured the database schema
# for real-time.
```

There is a template of the init file supplied in the /bgsys/drivers/ppcfloor/bin directory. The name of the file is bgpmaster.init.tpl.

3.1.2 BGPMaster command

The **bgpmaster** command is used to start, stop, or restart BGPMaster or any of the individual servers that it monitors. This command can also be used to check the status of the BGPMaster daemon and the individual servers that it is monitoring. You must run the command from the /bgsys/driver/ppcfloor/bin directory.

Syntax for the **bgpmaster** command is:

```
bgpmaster [options] <command>
```

Table 3-1 lists the required <command> parameter options.

Table 3-1 Command parameters for the **bgpmaster** command

Command	Syntax	Description
start	bgpmaster start <server-name>	If start is the only parameter supplied, the BGP Master daemon is started along with any of the servers specified in the bgpmaster.init file. If a server-name is specified, and that server is not already running, it will be started.
stop	bgpmaster stop <server-name>	If stop is the only parameter supplied, the BGP Master daemon is stopped along with any of the servers it has started. If a server-name is specified, and that server is running, it will be stopped.
restart	bgpmaster restart <server-name>	If the restart parameter is specified by itself BGP Master, and any of the servers it started, will be stopped and then restarted. If a server-name is specified, that server is stopped and restarted.
reload	bgpmaster reload	Tells the BGP Master daemon to reload and process the bgpmaster.init file.
status	bgpmaster status	Reports whether the BGP Master daemon is started. If it is the status of the server specified in the bgpmaster.init is returned.

Table 3-2 lists the available options for the **bgpmaster** command.

Table 3-2 Options for the **bgpmaster** command

Option	Syntax	Description
--autorestart	<y or n>	Determines if bgpmaster restarts failed servers automatically. If n is specified, a failed server must be restarted manually.
--host	<hostname or IP address>	Specifies the host name or IP address of the bgpmaster daemon. Default is 127.0.0.1
--port	<IP port>	Specifies the port on which the bgpmaster daemon listens.
--help	none	Provides help text

Table 3-3 provides an additional list of options that are used when the start and restart commands are used. These options are ignored if they are used with the stop, reload or status command.

Table 3-3 Start and restart options

Option	Syntax	Description
--binpath	<directory>	Specifies the directory that contains the BGPMaster program. The default is /bgsys/drivers/ppcfloor/bin.
--configpath	<directory>	Specifies the path to the directory that contains the configuration file. The default is /bgsys/local/etc.
--useDatabase	<machine serial number>	Specifies the machine serial number for the system being managed by the control system. In most cases it will be <i>BGP</i> . If a value is not specified the default value is the system name in the db.properties file. Used to separate log files for different Blue Gene systems.
--db2profile	<db2profile file name>	Specifies the db2profile to source for connecting to the database. Default is <i>db2profile</i> .
--dbproperties	<dbproperties file name>	Specifies a dbproperties to source for connecting to the database. Default is <i>db.properties</i> .
--baseLogdir	<directory>	Names the path to the directory where log files will be created. The default BGPMaster log file will be /baseLogdir/machine SN /<hostname>-bgpmaster-<date:time>.log.
--loglink	<symlink name>	Specifies the name of the symlink to the latest (current) log file. The default is /baseLogdir/machine SN /<hostname>-bgpmaster-current.log
--config	<configuration file name>	Specifies the name of the bgpmaster configuration file. The default is bgpmaster.init.

3.1.3 BGPMaster's Navigator interface

As shown in Figure 3-1, you are able to stop, start, and restart each of the control system daemons (with the exception of the navigator_server) from the Navigator interface. When the Navigator interface is used to start or restart a component, the daemon runs under the profile that originally started BGPMaster. BGPMaster itself must be started and stopped from the command line.

BGP Master						
Name	Status	PID	Action	Auto Start	Start Command	Status Message
mcserver	Started	8097	Stop Restart	Yes	startmcserver	
mmcs_server	Started	8101	Stop Restart	Yes	startmmcs	
mpirund	Started	8107	Stop Restart	Yes	startmpirund	
navigator_server	Started	8111	Stop	Yes	startnavigator	
realtime_server	Stopped		Start		startrealtime	need Administrator to restart. Down for 31962 seconds
submit_server	Started	8127	Stop Restart	Yes	startsubmitserver	

Figure 3-1 BGPMaster's Navigator interface

3.2 mcServer

Important: The MMCS server relies on the mcServer to monitor the status of the hardware. If the mcServer is stopped for an extended period, the mmcs_server will fail. In the default configuration, BGPMaster tries to restart both servers if the **stop** command has not been issued.

mcServer is a wrapper around the Machine Controller (mc). Machine Controller is a library of functions providing low level control system access to the hardware. mcServer wraps this library to provide a server and ports to connect to. mcServer handles arbitrations and controls access to the hardware to prevent simultaneous conflicting operations. It also provides Target Sets which gives the client a way to refer to a collection of hardware with a single name or handle. Another function of mcServer is to provide some event handling. Clients can register to receive RAS and console events for a particular location or for all events.

Evolution Note: In Blue Gene/L, this low-level control was provided by the idoproxy.

Figure 3-2 illustrates how mcServer provides a layer of arbitration between the hardware and higher level clients. Examples of clients would be MMCS, diagnostics, or service actions.

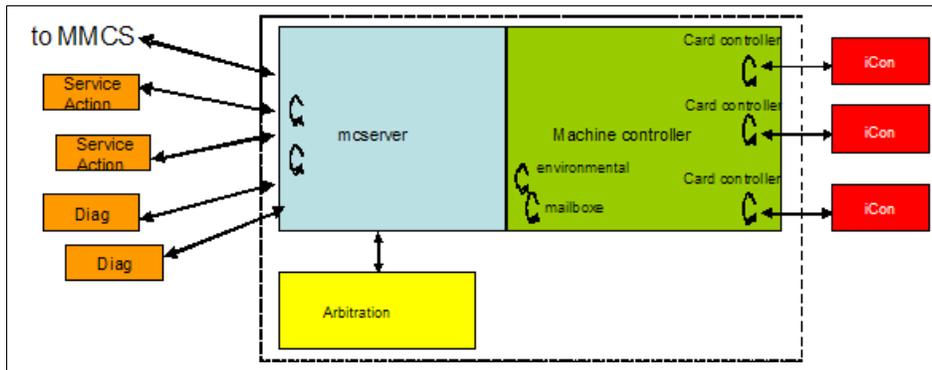


Figure 3-2 mcServer layer

3.2.1 Starting mcServer

Under normal circumstances, BGPMaster starts mcServer. However, there can be occasions that might require a manual start. In these circumstances, you can change to the directory `/bgsys/local/etc`. Check to make sure that the `mcserver_params.xml` file exists and the data it contains is correct. Example 3-2 shows a sample `mcserver_params.xml` file.

Example 3-2 Sample `mcserver_params.xml` file

```
<MCServerParams
  clientPort="1206" servletPort="1207"
  machineType="0"
  hostName="localhost"
  autoinit="false" skeletonLocs="false"
  rackRows="1" rackColumns="1">
<Subnet netMask = "255.255.0.0" addressBase = "10.0.100.0" addressCount="255" />
<_hwOptions value="dd2=false" />
<_hwOptions value="debug=2" />
</MCServerParams>
```

Most of the parameters in the `mcserver_params.xml` file are created when the system is installed. For example, the `machineType="0"` indicates that it is a standard Blue Gene/P machine. It is important to verify that the `rackRows` and `rackColumns` values are correct for your environment and that the default ports of 1206 and 1207 are not already in use.

Table 3-4 provides a list of the debug levels that can be supplied as an option when the mcServer is started.

Table 3-4 Debug Levels

Debug Level	Explanation
0	Silent mode
1	Default mode, minimal messages
2	Print message for each command received by the server, thread create/terminate
5	Print detailed event messages, events received (delivered to listener) and dropped (no listener)

Debug Level	Explanation
10	Dump XML request/reply objects for most commands
19	Dump XML request/reply objects for all commands, even the ones that return large amounts of data
>19	Detailed debug information concerning mcServer internal operation.

After verifying that the file contains the correct information for your environment, issue the following command (assuming that you have at least bgpadmin authority):

```
bgpadmin@servicenode:/bgsys/driver/ppcfloor/bin>./bgpmaster start mcserver
```

Alternatively, if you need to run without BGPMaster you can use this command:

```
bgpadmin@servicenode:/bgsys/local/etc>./mcserver mcserver_params.xml
```

The last option is to start mcServer from the Navigator BGP Master interface. In the row that begins with *mcserver* click the Start button in the Action column.

3.2.2 Stopping mcServer

You can stop mcServer in several ways. First, you can simply end it by stopping BGPMaster, which brings down all of the servers.

You can also end the individual server using the following command in `/bgsys/drivers/ppcfloor/bin`:

```
./bgpmaster stop mcserver
```

Another option is to click **Stop** next to the mcserver entry on the BG/P Master page in the Navigator.

3.2.3 Logging

mcServer writes its logs to the `/bgsys/logs/BGP` directory. The current server instance log file is linked to `xxxxx-mcserver-current.log` (where `xxxxx` is the host name of your service node). A new log is created each time mcServer is started. Historical log files are named using the following syntax `xxxxx-mcserver-yyyy-mmdd-hh:mm:ss.log`. You can also access these logs through the Navigator by clicking the System Logs link in the Navigation pane (Figure 3-3). You can select current or historical logs from this interface by clicking the Current, All or Old tab.

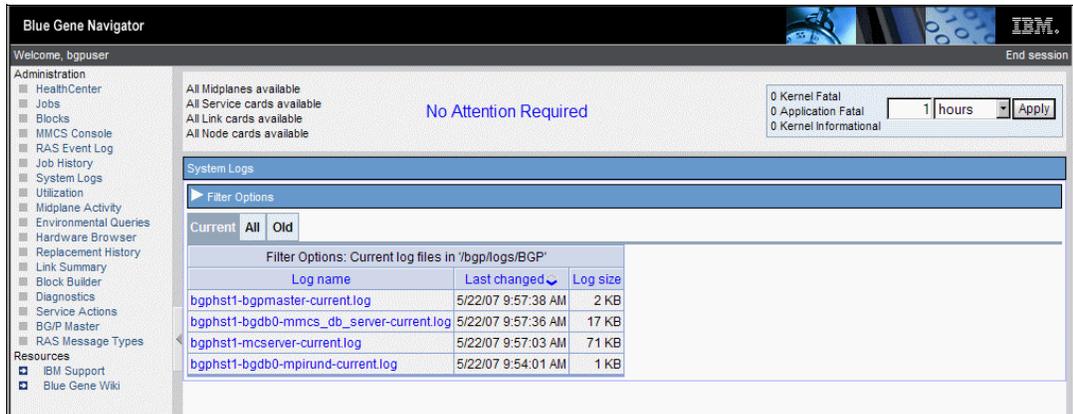


Figure 3-3 System Logs

3.3 Midplane Management Control System

The Midplane Management Control System (MMCS) is used to configure and allocate blocks of Compute and I/O nodes and to run programs on the Blue Gene/P system. Referred to as the *high-level control system*, MMCS server runs on the Service Node and provides an interface to the hardware level system components.

MMCS can be accessed directly using the MMCS console, Blue Gene Navigator, or through the various scheduler programs (using the scheduler APIs). The MMCS processes that run on the Service Node also provide the polling services for the Environmental Monitor. By default the Environmental Monitor is active any time the MMCS server is running. For more specific information about MMCS, see Chapter 5, “Midplane Management Control System” on page 91.

3.3.1 Starting MMCS server

Normally, the BGPMaster processes start or restart the MMCS server when necessary. As with the mcServer processes, there might be an occasion where you need to start the MMCS server manually. Keep in mind that the MMCS server relies on mcServer to communicate with the hardware. Verify that the mcServer is running before starting the MMCS server.

The syntax to start the MMCS server using BGPMaster is:

```
bgpadmin@servicenode:/bgsys/driver/ppcfloor/bin>./bgpmaster start mmcs_server
```

You can also use the `mmcs_db_server` command in the `/bgsys/driver/ppcfloor/bin` directory. The drawback to starting the server this way is that no log files are created.

The last option is to start the MMCS server from the Navigator. Click **BGPMaster** in the navigation pane. Then, click the **Start** or **Restart** button that is associated with the MMCS server (`mmcs_server`).

3.3.2 Stopping MMCS server

You can use several methods to stop the MMCS server. First, you can end it by stopping BGPMaster, which brings down all of the servers that were started when the BGPMaster process was initialized. You use the following command:

```
bgpadmin@servicenode:/bgsys/driver/ppcfloor/bin>./bgpmaster stop
```

You can also specifically end the MMCS server through BGPMaster using the following command in `/bgsys/drivers/ppcfloor/bin`:

```
./bgpmaster stop mmcs_server
```

Another option is to click the **Stop** button next to the `mmcs_server` entry on the BG/P Master page in the Navigator.

3.3.3 Logging

Starting the MMCS server through BGPMaster or using the `startmmcs` command creates a log file using the syntax `systemname-mmcs_db_server-yyyy-mmdd-hh:mm:ss.log`. It also creates `systemname-bgdb0-mmcs_db_server-current.log` as a symlink to the log file. The link is updated to link to the new file each time the server is started. If the `mmcs_db_server` command is used to start the MMCS server, no log file is generated.

Log files can be viewed from the command line or with the Navigator. The MMCS server logs are stored in the `/bgsys/logs/BGP` directory.

3.4 The mpirun daemon

The mpirun daemon is a general purpose job launcher that operates with a client/server relationship. Clients, either on the Blue Gene/P Front End Node or the Service Node, submit requests to the server running on the Service Node. The server process running in BGPMaster is the mpirun daemon (`mpirund`), which listens for these incoming mpirun requests. The mpirun daemon listens for incoming requests on port 9874.

Figure 3-4 depicts how mpirun interacts with the rest of the control system.

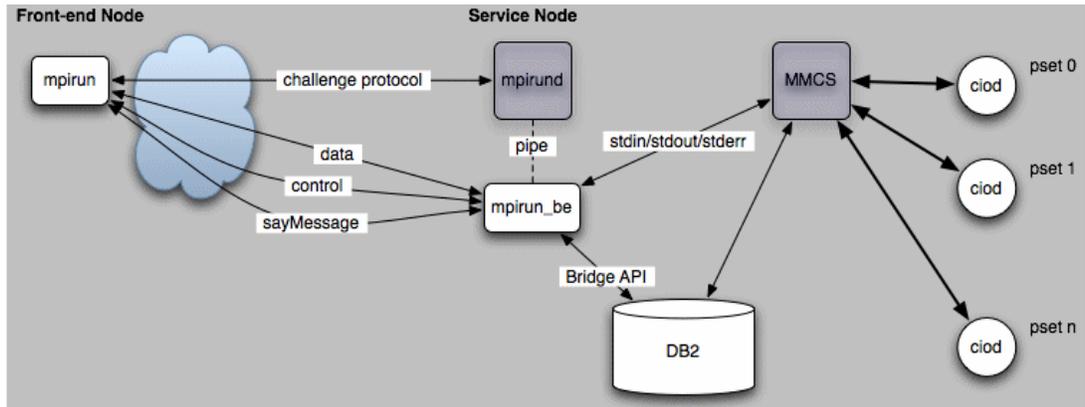


Figure 3-4 The mpirun daemon

Challenge protocol

The protocol used to authenticate the mpirun Front End Node when connecting to the mpirun daemon on the service node is a challenge/response protocol. This protocol uses a shared secret to create a hash of a random number, thereby verifying the mpirun Front End Node has access to the secret. To protect the secret, it is stored in a configuration file only accessible by the `bgpadmin` user on the service node and by a special `mpirun` user on the Front End Node. The front end `mpirun` binary has its `setuid` flag enabled so it can change its UID to match the `mpirun` user and to read the config file to access the secret.

Front-end and back-end mpirun

Mpirun is comprised of two programs: front-end mpirun and back-end mpirun. The front-end program is simply called *mpirun*. The back-end program is referred to as *mpirun_be*. Users only come into contact with mpirun on the Front End Nodes.

Mpirun can run on any machine in the Blue Gene/P environment. There is no prerequisite for a database client to be installed on these machines, because the front-end program does not access the database. However, *mpirun_be* does access the database, so the database client must be installed on the Service Node.

When the authentication is complete the transaction is handed off to the *mpirun_be* process. The *mpirun_be* process initiates a callback to the client based on the information passed from the daemon. The client and *mpirun_be* negotiate the opening of three more connections to complete the mpirun client request.

Bridge APIs

The *mpirun_be* updates the Blue Gene/P database with the information about the job using the Bridge APIs. For more information about this topic, see 1.2.13, “Bridge APIs” on page 13.

MMCS Server

The MMCS server monitors the database for updates to various tables (block and job tables). For more information, see 3.3, “Midplane Management Control System” on page 70.

3.4.1 Configuring mpirun

There are three significant files used by the mpirun:

<code>db.properties</code>	Contains information about the Blue Gene/P database.
<code>bridge.config</code>	Provides the default locations for the I/O node and compute node images to be used when allocating blocks.
<code>mpirun.cfg</code>	Contains the shared secret that is used for the challenge authentication between mpirun and mpirund.

The user that starts BGPMaster must have read access to all of the files.

db.properties file

The *mpirun_be* process uses Bridge APIs to make updates to the Blue Gene/P database concerning the mpirun requests received. The `db.properties` file contains the information required for mpirun to access the database. By default mpirun looks in the `/bgsys/local/etc` directory on the Service Node for the file when the daemon is starting.

Tip: There is a template of the `db.properties` file in the `/bgsys/drivers/ppcfloor/bin` directory (named `db.properties.tpl`). If you do not see the file in the `/bgsys/local/etc` directory, you can copy the template into the directory. When you copy the file, rename it *db.properties* (remove the `.tpl` extension).

Example 3-3 shows the section of the `db.properties` file that is used by the mpirun.

Example 3-3 The db.properties.tpl file

```
database_name=bgdb0
database_user=bgpsysdb
database_password=YourDBPassword
database_schema_name=bgpsysdb
```

```

system=BGP
min_pool_connections=1
mmcs_envs_purge_months=3
mmcs_boot_options=

```

bridge.config file

The bridge.config file contains the path to the images that are used for the I/O nodes and compute nodes to boot. Example 3-4 contains an example of the default file. This file should exist on the Service Node in the /bgsys/local/etc directory.

Mpirun uses the BGP_DEFAULT_CWD entry (last line of the example) to determine what the job's default **cwd** should be if the user fails to give the **-cwd** argument. There is a special case for BGP_DEFAULT_CWD=\$PWD, mpirun expands that variable to the **cwd** where mpirun was launched. As an example, a system administrator might want to change this to something similar to BGP_DEFAULT_CWD=/gpfs to force jobs without an explicit **cwd** to run in their GPFS™ file system.

Example 3-4 The bridge.config file

```

BGP_MACHINE_SN      BGP
BGP_MLOADER_IMAGE  /bgsys/drivers/ppcfloor/boot/uloader
BGP_CNLOAD_IMAGE
/bgsys/drivers/ppcfloor/boot/cns,/bgsys/drivers/ppcfloor/boot/cnk
BGP_ILOAD_IMAGE
/bgsys/drivers/ppcfloor/boot/cns,/bgsys/drivers/ppcfloor/boot/linux,/bgsys/drivers
/ppcfloor/boot/ramdisk
BGP_BOOT_OPTIONS
BGP_DEFAULT_CWD    /bgusr

```

mpirun.cfg file

The mpirun.cfg file contains the shared secret that is used by the mpirun daemon in the authentication process. Example 3-5 shows the format of the mpirun.cfg file. This file needs to exist on the Service Node and any Front End Nodes that submit jobs using the mpirun command. Regardless of the system, the files need to match exactly for the authentication process to work. The mpirun.cfg file is in the bgsys/local/etc/ directory. Only bgpadmin and the special mpirun user needs to have access to the file.

Example 3-5 The mpirun.cfg file

```

CHALLENGE_PASSWORD=BGPmpirunPasswd

```

Tip: If you need a copy of the mpirun.cfg file, there is a template in the /bgsys/drivers/ppcfloor/bin directory.

3.4.2 Environment variables

The backend process uses two environmentals: DB_PROPERTIES and BRIDGE_CONFIG_FILE. If the DB_PROPERTIES variable is not set, the backend process looks for a db.properties file in the /bgsys/local/etc directory. If the BRIDGE_CONFIG_FILE variable is not set, the backend aborts.

Both environmentals are validated by mpirund upon startup because they are always passed to mpirund through the startmpirund script. If either of these config files does not exist, mpirund will fail to start.

3.4.3 Front End Node

There must be an `mpirun.cfg` file configured on the Front End Node in the `/bgsys/local/etc` directory. This file contains the password used to authenticate the initial mpirun communications and access should be restricted to `bgpadmin` and the `mpirun` user. As mentioned previously, the contents of this file must match the contents of the file on the Service Node exactly.

The Service Node must also be able to resolve the host name of the Front End Node either through its own host table or from a DNS server. When the initial connection is made the Front End Node provides the Service Node with its host name. In turn the Service Node uses the host name in its callback process to open new connections back to the Front End Node.

3.4.4 Starting the mpirun daemon

By default, the mpirun daemon starts automatically when you start BGP Master. Starting the server this way allows the BGP Master process to monitor the server and to restart the daemon if it should fail.

If the mpirun daemon has stopped for some reason, you can start it manually in one of two ways. In the `/bgsys/drivers/ppcfloor/bin` directory, you can issue the following command:

```
bgpadmin@servicenode:/bgsys/driver/ppcfloor/bin>./bgpmaster start mpirund
```

From the Navigator, on the BG/P Master page, you can click the **Start** (or **Restart**) button associated with `mpirund`.

3.4.5 Stopping the mpirun daemon

You can stop `mpirund` in several ways. First, you can end it by stopping BGP Master, which brings down all of the servers, using the following command:

```
bgpadmin@servicenode:/bgsys/driver/ppcfloor/bin>./bgpmaster stop
```

You can also end the individual server through BGP Master using the following command in `/bgsys/drivers/ppcfloor/bin`:

```
./bgpmaster stop mpirund
```

Another option is to click **Stop** next to the `mpirund` entry on the BG/P Master page in the Navigator.

3.4.6 Logging

The default location for the mpirun daemon logs is `/bgsys/logs/BGP`. The naming syntax used when creating the log files is `<hostname>-mpirund-yyyy-mmdd-hh:mm:ss.log`. When the server is started a link to the current log (in the same directory) is created. This link file is named `<hostname>-bgdb0-mpirund-current.log`. This current file can be monitored from the command line or Navigator.

3.5 Navigator server

Blue Gene Navigator is a full featured Web-based administration tool. User can also use it to monitor jobs, blocks, current usage, and the availability of the Blue Gene hardware. The `navigator_server` process powers the Navigator Web pages. When the server is running, you

can access the secure Web pages at <https://mybluegenep:32072/BlueGeneNavigator>. There are some configuration steps that are required to set up the Web pages. For more information about setup, see Chapter 2, “Blue Gene Navigator” on page 17.

3.5.1 Starting the navigator_server process

The Blue Gene Navigator (navigator_server) is set to start automatically when BGPMaster is started. In the event that you need to start the server manually and that you want BGPMaster to monitor it, use the following command:

```
$ /bgsys/driver/ppcfloor/bin/bgpmaster start navigator_server
```

3.5.2 Stopping the navigator_server

If the Navigator was started with BGPMaster, it ends if you stop the BGPMaster process. You can also end the Navigator server individually by issuing the following command:

```
$ /bgsys/driver/ppcfloor/bin/bgpmaster stop navigator_server
```

You can stop the navigator_server process from the Navigator itself by clicking **Stop** in the mmcs_server's row. At that point, you have to restart the server from the command line.

3.5.3 Logging

The Navigator writes logs to the /bgsys/logs/BGP directory. The file <hostname>-bgdb0-navigator-current.log is a link to the log of the Navigator instance that is currently running. The actual files that the logs are being written to use the naming syntax <hostname>-navigator-yyyy-mmdd-hh:mm:ss.log. You can view or tail the files from the command line. You can also view or monitor the logs from the Navigator by clicking **System Logs** in the navigation pane.

3.6 Real-time

The Real-time server provides schedulers with changes to the database as certain events occur. These events cause updates in the database, for example, when a partition is freed. Triggers in the database update the state on the Real-time server. In turn, the server sends the updates to the scheduler based on the scheduler's callback requests.

When the scheduler is initially started the *state* of the system is acquired from the database using the Bridge APIs (*rm_get_**). This provides the scheduler with a snapshot to use as a starting point. As subsequent events occur the scheduler is updated with the changes. You can find more detailed information in *IBM System Blue Gene Solution: Blue Gene/P Application Development*, SG24-7287.

Evolution Note: In Blue Gene/L when schedulers wanted an update, they had to poll the database using the Bridge APIs. This constant polling could have a negative impact on performance.

Blue Gene/P database tables of interest to the schedulers have a sequence identification (seqid) field in them. As events enter the database, the seqid increments. The event with largest value in the seqid field is always the most recent event in that table.

Database tables that contain the sequence identification field are:

TBGPBLOCK	blocks
TBGPJOB	jobs
TBGP MIDPLANE	base partitions
TBGP NODECARD	node cards
TBGP SWITCH	switches

3.6.1 Configuring Real-time

You set up the Real-time server by running the `configureRealtime.pl` program. Running the program sets up the database schema for the new Real-time server code. You need to run the program prior to starting the Real-time server. Normally, the server runs using the `bgpsysdb` db2 profile. You need to verify that this user can read the `db.properties` file and can write to the log directory (`/bgsys/logs/BGP` unless otherwise specified). There are three modes of operation for this program:

<code>install</code>	Adds any stored procedure that is not defined or alters any stored procedure that is defined, and adds any trigger that is not defined.
<code>update</code>	If any stored procedure or trigger is found it does the same as <code>install</code> , otherwise it does nothing.
<code>uninstall</code>	Removes all stored procedures and triggers.

For a new install, you typically run the following sequence of commands:

```
$ . ~/bgpsysdb/sql1lib/db2profile
$ cd /bgsys/drivers/ppcfloor/schema
$ ../bin/logOutput --svcname configureRealtime ./configureRealtime.pl install
```

When doing an update to the driver (release), follow this sequence:

```
$ . ~/bgpsysdb/sql1lib/db2profile
$ cd /bgsys/drivers/ppcfloor/schema
$ ../bin/logOutput --svcname configureRealtime ./configureRealtime.pl
```

The default mode is `update`.

Other options available for `configureRealtime.pl` are:

<code>--configpath</code>	Overrides the default configuration path of <code>/bgsys/local/etc</code> .
<code>--dbproperties</code>	Overrides the default <code>db.properties</code> file of <code><configpath>/db.properties</code> .
<code>--properties</code>	Overrides the default <code>configureRealtime.properties</code> files of <code><exepath>/configureRealtime.properties</code> .
<code>--mode</code>	Overrides the default mode of <code>update</code> .
<code>--help</code>	Prints help text and exits.

3.6.2 Starting the realtime_server process

By default the Real-time server is not set to start automatically in the `bgpmaster.init` file. To set the server to start with `BGP Master`, uncomment the following line in the init file:

```
#start realtime_server
```

To start the Real-time server manually, run one of the following commands:

```
$ /bgsys/driver/ppcfloor/bin/bgpmaster start realtime_server
```

or

```
$ /bgsys/local/etc/startrealtime
```

You can also start the Real-time server through Blue Gene Navigator. In the Navigation pane click the BG/P Master link. In the main content area, click **Start** to the right of the realtime_server.

3.6.3 Stopping the Real-time server

You can stop the Real-time server in one of three ways. First, you can end all of the BGPMaster monitored servers (and BGPMaster) by issuing the following command:

```
bgpadmin@servicenode:/bgsys/driver/ppcfloor/bin>./bgpmaster stop
```

Alternatively, you can stop just the Real-time server with the following command:

```
bgpadmin@servicenode:/bgsys/driver/ppcfloor/bin>./bgpmaster stop  
realtime_server
```

You can also click the **Stop** button that is associated with the realtime_server on the BG/P Master page in Navigator.



4

Blocks

Blocks, or partitions as they are also known, provide the means to identify all or part of the hardware on Blue Gene/P onto which jobs can be submitted. In this chapter, we explain blocks in more detail.

4.1 Blocks

Before you can submit any job to the Blue Gene core, you must create a block or partition in the database that defines the resources that are used for the job. The hardware and cabling of the Blue Gene/P core system determines which blocks you can create on your system. Block definitions are used to describe the Blue Gene/P resources that are used for running jobs. Blocks can be either *dynamic* or *static*. These block definitions are identified with a unique Block ID, which can be specified using up to 32 alphanumeric characters, dashes, and underscores.

Figure 4-1 shows how a block definition displays in the Navigator.

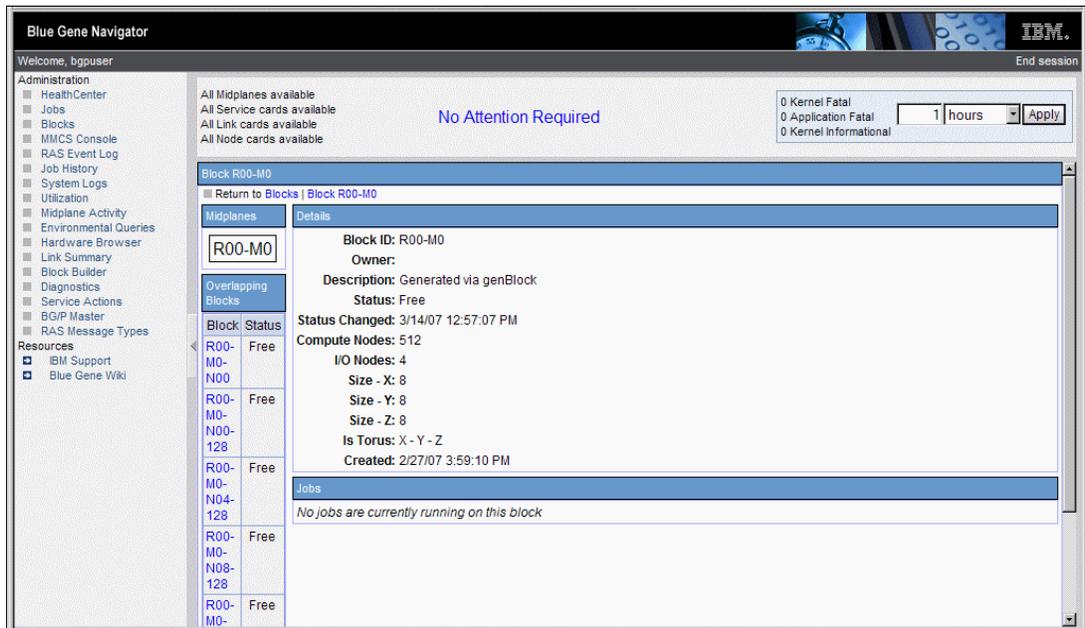


Figure 4-1 Block Details in the Navigator

You can obtain the same information from the database using the following command line:

```
db2 "select * from bgpblock where blockid = 'R00-M0'"
```

The DB2 query yields more information about the block, including the location of the compute node images, I/O node images, and the microloader.

The Base Partition of any Blue Gene/P system is a *midplane*. Midplanes are cabled together using Link Cards and data cables to create larger blocks. The size of the blocks that you can create and which midplanes you can use for the blocks are determined by the way your Blue Gene/P core is cabled. You can configure blocks that are created using one or more full midplanes as either a three-dimensional (3D) Mesh or a looping 3D Torus network. Later in the chapter, we discuss how to create blocks for multiple rack systems.

You can subdivide Base Partitions into small blocks. These blocks depend on the network fabric that is provided by the midplane. In addition, there are specific rules about creating these blocks. You can create Small Blocks with sizes of 16, 32, 64, 128, or 256 compute nodes. The number of small blocks that you create for a midplane depends on the number of I/O nodes in that midplane. Each block needs at least one I/O node. So, theoretically, you could subdivide a midplane that is fully populated with I/O nodes into 32 blocks, each with 16 compute nodes and one I/O node. Small Blocks can only use Mesh data connections, which

are actually hardwired into the Node Cards. As we mentioned previously, only blocks that consist of a single midplane or more can take advantage of the 3D Torus network.

You create dynamic blocks using a job submission program, such as LoadLeveler or mpirun. The user that submits the job tells the submission program details about the Blue Gene/P resources, such as the number of processors that are needed for the job, and the size of processor set (the number of I/O nodes) to create for the job, the microloader, Compute Node images and I/O node images to use. The job submission program creates the block based on this information using the MMCS Bridge APIs and any of the Blue Gene/P resources that are available at the time to run the job. The user cannot choose which resources are used. When the job has completed, the block is freed and destroyed and no longer exists in the MMCS database.

The system administrator creates static blocks, and they remain in the MMCS database to be reused when needed. These block definitions also include the specific hardware resources on the Blue Gene Core, the pset size of the block, and the microloader, I/O node images, and compute node images. This allows the system administrator to assign a block for a particular user to use specific core resources. When users submit their jobs, they only need to specify which static block they are going to use to execute the job.

4.2 Blue Gene Navigator

There are two areas of Blue Gene Navigator that pertain to blocks, In the navigation pane, they are labeled *Blocks* and *Block Builder*. Selecting **Blocks** link opens the page shown in Figure 4-2. This page is available in both the administrator's and the user's version of the Navigator. You can chose to view blocks that are in use, available, or free. The default is to show the blocks that are available for the entire system (all Blue Gene/P hardware), but you can filter the view (by expanding the Filter Options) to a single midplane by selecting that specific midplane in the Filter Options. You can also filter by the mode or by High Throughput Computing (HTC) pool.

Block Summary								
Filter Options								
Mode: <input checked="" type="checkbox"/> HPC <input checked="" type="checkbox"/> HTC - DUAL <input checked="" type="checkbox"/> HTC - SMP <input checked="" type="checkbox"/> HTC - VN								
<input type="checkbox"/> HTC Pool: <input type="text"/>								
<input type="checkbox"/> Show blocks using midplane: <input type="text" value="R00-M0"/>								
<input type="button" value="Apply Filter"/>								
In Use Available Free								
Filter Options: All blocks in use								
Block	Owner	Status	Status Changed	Created	Size	Job Status	HTC Mode	HTC Pool
R00-M0-N01-J00	tmusta	Booting	5/29/08 2:48:15 PM	3/21/08 7:56:29 AM	16			
R00-M0-N07-J01_test	mklight	Booting	5/29/08 2:58:51 PM	5/22/08 2:41:26 PM	16			
R00-M0-N11-J00	mtn	Initialized	5/29/08 1:44:52 PM	3/21/08 7:57:56 AM	16	Running		
R00-M0-N15	faraja	Initialized	5/29/08 1:45:35 PM	3/20/08 6:11:41 PM	32			
_DIAGS_R20-M1	clappi	Booting	5/29/08 2:57:33 PM	5/29/08 2:57:24 PM	512			

Figure 4-2 Block Summary page

Notice that the block names are links. Clicking on one of the links will open the Block Details page shown in Figure 4-3. This page contains information about the block such as its size, the number of I/O nodes and whether it is a torus. Blocks that use all or some of the hardware included in this block are listed on the left side of the page. Blocks that consist of a midplane (512 nodes) or more are also displayed using the Block Visualizer. The Block Visualizer

shows a three dimensional view of the entire system. Midplanes used in a block, pass through midplanes and midplanes that are not involved in the block are all colored differently.

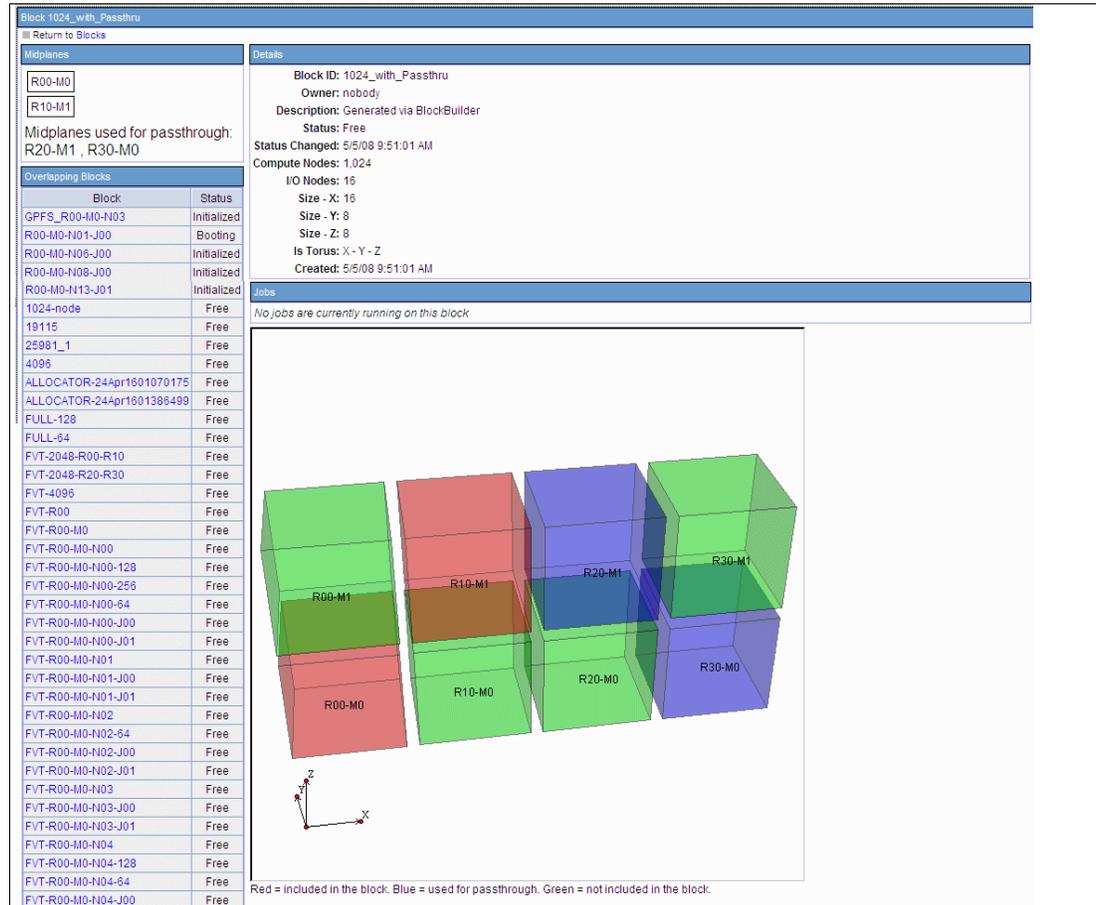


Figure 4-3 Block Details page

Figure 4-4 show the block details of a block that has been booted in HTC SMP mode. Notice that in the Jobs section it shows the number of jobs currently running and the total number of jobs that have completed. In addition to the usual information returned you also will see sections containing Hardware Used and Failed Nodes.

The Blue Gene/P version of HTC does not try to recover nodes that have failed. Unlike a block that is booted in HPC mode, a single failed node will not cause the entire block to become unavailable. The failed nodes are removed from the pool of available nodes and the remaining jobs continue to run.

Block	Status
10081	Free
2048-R00-R01	Free
2048-R00-R 0	Free
2 405	Free
3 377	Free
4096	Free
4096p	Free
5023	Free
BCU004	Free
G ² FS_1C95	Free
G ² FS_4C95-32	Free
G ² FS_R0	Free
G ² FS_R00	Free
G ² FS_R00-MC	Free
G ² FS_R00-MC-16	Free
G ² FS_R00-MC-64	Free
G ² FS_R00 MC IOR	Free
GSLFU_L	Free
MGV_PT	Free
MPIRUN-11Sep0912372229	Free
MPIRUN-14Nov0542030113	Free
MPIRUN-14Nov0554247985	Free
MPIRUN-14Nov0558473222	Free
MPIRUN-14Nov0601021442	Free
MPIRUN-14Nov0607095013	Free
MPIRUN-14Nov0645199103	Free
MPIRUN-15Dec0108311761	Free
MPIRUN-28Aug0135095832	Free

Block ID: R00-M0-N14
Owner: meger
Description: Generated via genSmellBlock
Status: Initialized
Status Changed: 3/6/08 9:34:33 AM
HTC Mode: SMP
HTC Pool: R00-M0-N14
Compute Nodes: 32
IO Nodes: 1
Size - X: 4
Size - Y: 4
Size - Z: 2
Is Torus: No Torus
Options: s
Created: 8/20/07 1:59:05 PM
Error Text:

Hardware Used

Pset	I/O Node	Compute Nodes
1	R00-M0-N14-J0C	32

Jobs

2 HTC jobs are currently running on this block. [Show current jobs on this block.](#)
2 HTC jobs ran on this block. [Show previous jobs for this block.](#)

Failed Nodes

4 Nodes Failed

Location	Reason	Job
R00-M0-N14-.04	Kernel 'ata: RAS event	
R00-M0-N14-.05	Kernel 'ata: RAS event	
R00-M0-N14-.05	Kill job timed out	
R00-M0-N14-.05	Kill job timed out	4941

Figure 4-4 Block details

Figure 4-5 shows the Block Builder page. You can navigate to this page by selecting the Block Builder option in the navigation pane of the Navigator interface.

Figure 4-5 Block Builder

Just below the Block Builder title bar there is one or more block diagrams that contain a representation of all the midplanes in your system. Figure 4-6 depicts a system that consists of two racks. To create a block that consists of at least one base partition (midplane) click the **Add** button for each of the midplanes that you want included in the block. If you need to use a midplane’s link card to complete the block’s torus, without including that midplane’s compute

or I/O nodes in the block, click the **Passthru** button for the corresponding midplane. Notice that as you choose the midplanes to be added to the block, the midplane label is outlined in red. If you choose to use a midplane as a Passthru the midplane label is outlined in blue. Additionally, once you chose either the Passthru option, or add more than one midplane to your block, the small block options (the last row of options in Figure 4-5) are removed from view.

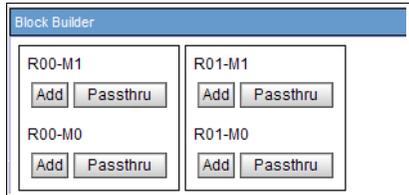


Figure 4-6 Midplane selection

To create a block consisting of one or more midplanes using Blue Gene Navigator follow these steps:

1. Select the midplane (or midplanes) that you want to include in your block.
2. Provide a name for the block. The Block Name can be up to 32 characters in length and must contain only alphanumeric characters, dashes, and underscores.
3. Next, select the ratio. This ratio reflects the number of I/O nodes to compute nodes. You can compute the ratio by dividing the number of compute nodes in the block by the number of I/O nodes that are included.
4. Finally, select the Configuration, which will be either Torus or Mesh, and click **Create Block**.

Using the Navigator in Blue Gene/P, you can also create small blocks (sub-midplane blocks) from the Web interface. To create a small block follow these steps:

1. First you need to select the midplane on which you want to create the block. Click **Add**, which outlines the selected midplane in red.
2. Supply a name for the block. Again, the name can contain up to 32 characters.
3. Next, select the ratio of I/O nodes to compute nodes. This can be determined by the number of compute nodes in the block divided by the number of I/O nodes.

Attention: Keep in mind that there must be at least one I/O node included in the block. The I/O node (or nodes) also has to be contained in one of the node card (or cards) that hold the compute nodes selected for the block.

The Configuration for a small block will always be a Mesh.

4. Select the Block Size from the pull-down choices:
 - 16-J00 - as you face the node card, this would be the I/O node and compute nodes on the left side of the node card
 - 16-J01 - as you face the node card, this would be the I/O node and compute nodes on the right side of the node card
 - 32 - all of the compute nodes and at least one of the I/O nodes on a node card
 - 64 - all of the compute nodes and at least one of the I/O nodes on two node cards. The starting node card must be N0, N2, N4, N6, N8, N10, N12, or N14.
 - 128 - all of the compute nodes and at least one I/O node on four node cards. The starting node card must be N0, N4, N8, or N12.

- 256 - all of the compute nodes and at least two I/O nodes in eight node cards. The starting node card must be N0 or N8. For example, if you are creating a block with 256 compute nodes and a ratio of 1:128 using the node cards in the front of the rack, your starting node card is N0. The I/O nodes for this ratio will be in the N0 and N4 node cards.
5. Choosing the Starting Node Card from the pull-down is the last step in the process. For the small blocks that consist of 16 or 32 compute nodes you can use any node card in the selected midplane provided it has the required I/O nodes. For the small blocks that are made up of 64 or more compute nodes you must use one of the node cards listed in the description in the previous step. After you select the starting node card, click **Create Small Block**. A message displays at the bottom of the windows letting you know if the block was created or if there was an error.

4.3 Midplane Management Control System

The other interface that is used to create static blocks is Midplane Management Control System (MMCS). In this section, we cover only the commands that relate to creating blocks. Chapter 5, “Midplane Management Control System” on page 91 goes into more detail about the rest of the supported MMCS commands. The supported set of commands for generating blocks are:

- ▶ genblock
- ▶ genblocks
- ▶ gensmallblock
- ▶ genBPblock
- ▶ genfullblock

Attention: Block names (referred to as <blockid> in the following sections) can be up to 32 alphanumeric characters, dashes, and underscores.

genblock

The **genblock** command is used to generate a block that consists of one base partition. See Figure 4-7. The syntax for the MMCS command follows:

```
genblock <blockid> <midplane> <machineSN> <psetsize>
```

The parameter's definitions are as follows:

<blockid>	The name of the block to be created in the BGPBlock table.
<midplane>	The POSINMACHINE (for example, R00-M1) value from the BGPMidplane table.
<machineSN>	The serial number defined in the BGP MACHINE table (usually BGP).
<psetsize>	The ratio of compute nodes to each I/O node (depends on the hardware configuration).

The following command creates a block named Example_1 that consists of a base partition using the hardware resources from R02-M0 (see Figure 4-7). The psetsize suggests that the rack has only eight I/O nodes per midplane.

```
mmcs$ genblock Example_1 R02-M0 BGP 64
```

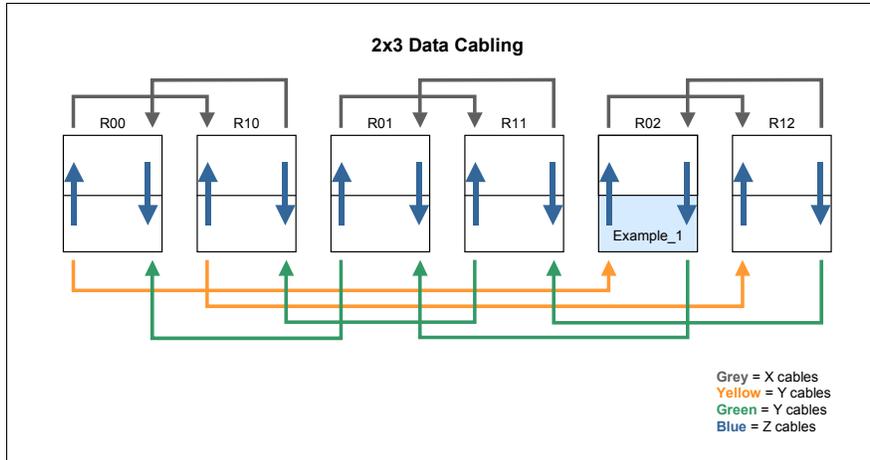


Figure 4-7 The genblock command creates a single Base Partition block

genblocks

The **genblocks** command generates a block for each base partition on the system. The syntax for the MMCS command is shown here:

```
mmcs$ genblocks <blockidprefix> <machineSN> <psetSize>
```

The parameters of the GENBLOCKS command are:

- <blockidprefix> The BLOCKID created in the BGPBLOCK table with a combination of prefix (if specified) and position of the midplane.
- <machineSN> The serial number defined in the BGPMACHINE table (usually BGP).
- <psetSize> The ratio of compute nodes to one I/O node (depends on the hardware configuration).

The following command creates a block for each Midplane on the system with a prefix of Ex_ and the midplane appended to the end (see Figure 4-8). The psetSize suggests that this is an I/O rich rack (every node card has two I/O cards installed).

```
genblocks Ex_ BGP 16
```

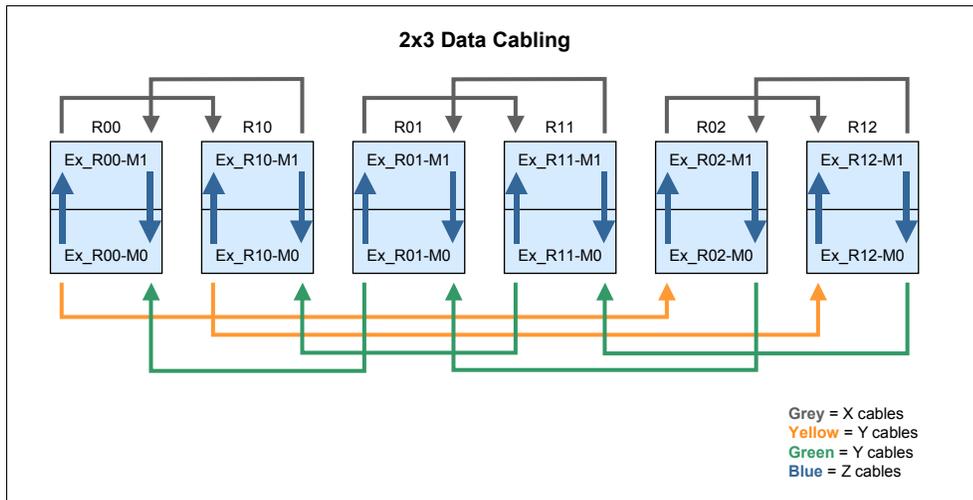


Figure 4-8 The genblock command creates a block from each midplane

gensmallblock

The **gensmallblock** command is used to generate a sub-base partition block. Small blocks can only be created as 32 node or 128 node blocks. The syntax for the MMCS command is:

```
gensmallblock <blockid> <midplane> <machineSN> <psetsize> <cnodes> <nodecard>
```

The parameters for the command are:

<blockid>	The name of the entry created in the BGPBLOCK table.
<midplane>	Specify the location of the midplane, for example R00-M0.
<machineSN>	The serial number defined in the BGPMACHINE table (usually BGP).
<psetsize>	The number of compute nodes to one I/O node (depends on the hardware configuration).
<cnodes>	The number of compute nodes for the block (16, 32, 64, 128, or 256 can be used depending on the number of I/O nodes in the midplane).
<nodecard>	The starting node card of the compute nodes.

Note: When the <cnodes> parameter is equal to 16, the <psetsize> can be used to pass in the I/O node number (either 0 or 1).

The following commands create 32 node blocks on Rack 00, Midplane 0, Node Card J104, J111, J117, J203, and J214 called R00-M0_Jxxx_32. See Figure 4-9 for hardware that is used for each of the blocks. The psetsize suggests that each node card has two I/O nodes installed with every node attached to the functional network.

```
gensmallblock R00-M0_N01_32 R00-M0 BGP 16 32 N01
gensmallblock R00-M0_N04_32 R00-M0 BGP 16 32 N04
gensmallblock R00-M0_N07_32 R00-M0 BGP 16 32 N07
gensmallblock R00-M0_N08_32 R00-M0 BGP 16 32 N08
gensmallblock R00-M0_N14_32 R00-M0 BGP 16 32 N14
```

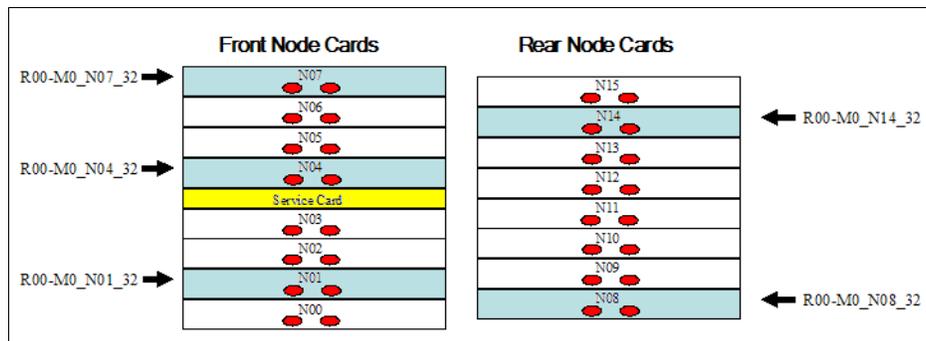


Figure 4-9 32 node small blocks

The following commands create four 128 node blocks on Rack 00 Midplane 0 (see Figure 4-10). The psetsize indicates that each node card has two I/O nodes installed, and all of the I/O nodes are attached to the functional network.

```
gensmallblock R00-M0_N00_128 R00-M0 BGP 16 128 N00
gensmallblock R00-M0_N04_128 R00-M0 BGP 16 128 N04
gensmallblock R00-M0_N08_128 R00-M0 BGP 16 128 N08
gensmallblock R00-M0_N12_128 R00-M0 BGP 16 128 N12
```

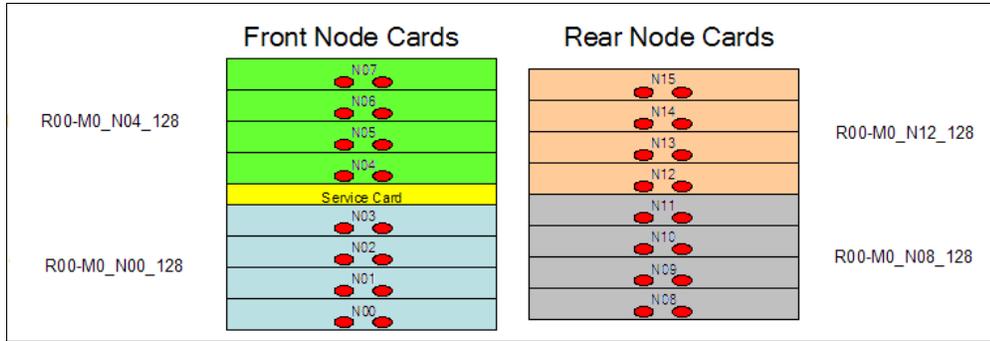


Figure 4-10 128 Compute Node block map

genBPblock

The **genBPblock** command is used to generate a block for a set of base partitions. There are some limitations for using the **genBPblock** command. This command cannot build blocks for midplanes that require pass-through or split cables.

When creating a block that does not require pass-through, the syntax for the MMCS command is:

```
genBPblock <blockid> <machineSN> <psetsize> <cornerBP> <xsize> <ysize> <zsize>
```

The parameters for the command are:

- <blockid> The name of the entry created in the BGPBLOCK table.
- <machineSN> The serial number defined in the BGPMACHINE table (usually BGP).
- <psetsize> The number of compute nodes to 1 I/O node (depends on the hardware configuration).
- <cornerBP> The starting midplane that is used to create a multiple base partition block; use the POSINMACHINE value from the BGPMIDPLANE table (Rxxx).
- <xsize> The number of midplanes to configure in the X dimension.
- <ysize> The number of midplanes to configure in the Y dimension.
- <zsize> The number of midplanes to configure in the Z dimension.

The following command creates a block for only the Midplanes on Rack 00 called Ex_R00 (see Figure 4-11). The psetsize indicates that each Node Card has only one I/O card installed and only one node attached to the functional network.

```
mmcs$ genBPblock Ex_R00 BGP 32 R00-M0 1 1 2
```

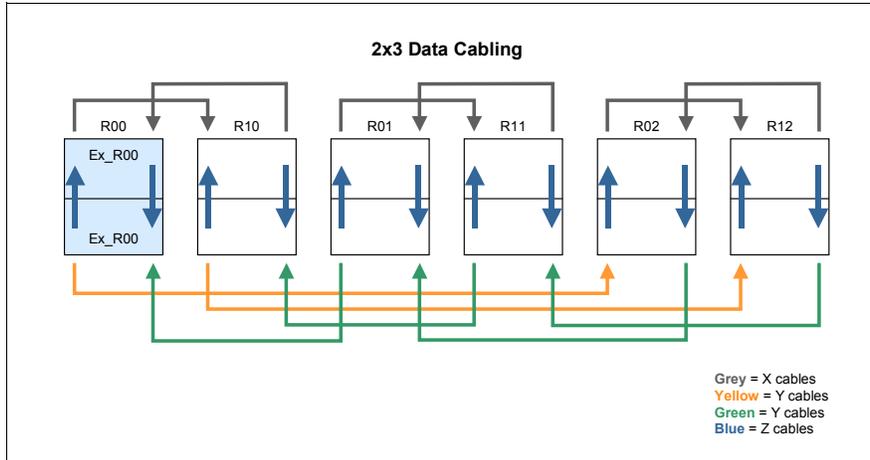


Figure 4-11 Creating a one rack block with genBPblock

Pass-through

Recall from earlier discussions about data cables that the data only flows in one direction. If you look at the data cable flow on Figure 4-11, notice that the Y dimension is comprised of green cables that connect the midplanes in sequence and yellow cables that skip a rack.

Figure 4-12 isolates the Y and Z cables to demonstrate the flow between the midplanes of R00 and R01. A block that consists of Rack 00 01 follows the data path of R00 → R02 → R01 → R00. R02 is used as a pass-through to forward the data on to R01. Because each Blue Gene/P system can be cabled differently, the **genBPblock** command is not able to create this block; it must be created using the Block Builder feature in Blue Gene Navigator where the pass-through midplane is configured.

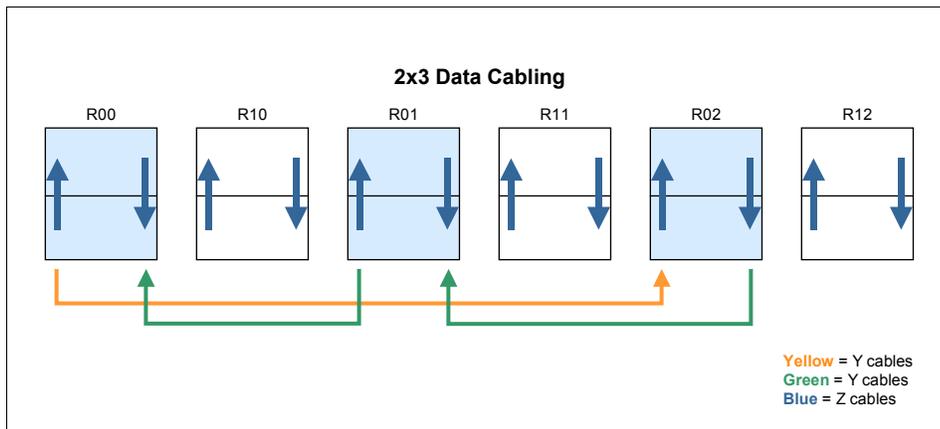


Figure 4-12 Data flow between R00 and R01

Split data flow

Split data flow happens in the X dimension on large systems. If you examine Figure 4-13, you can see the data flow for the X dimension on the bottom. Data flows from R0 → R2 → R3 → R1 → R0. If we create a block that consists of Row0 and Row1 (not using the split data flow), we must pass through Row2 and Row3 for Row0 to access Row1.

On the top of the diagram in Figure 4-13 is the data flow for Split X. If we create a block that consists of Row0 and Row1, we can use the Split X cable to go to Row1 and return on the X cable. This configuration can be different for each customer system. There is not a standard

way to configure the X dimension, so these blocks must be created in Blue Gene Navigator or defined in an XML file and imported into the MMCS database.

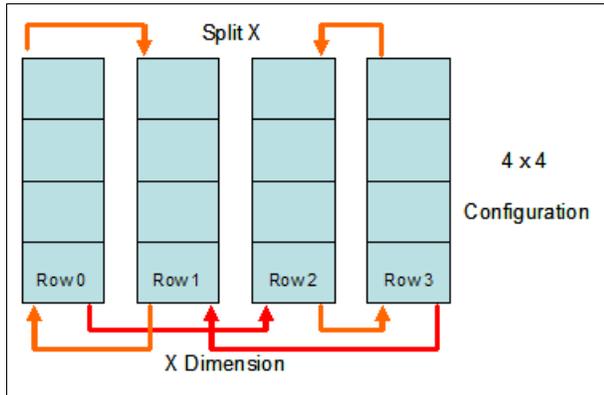


Figure 4-13 Data flow, X and split X

genfullblock

The **genfullblock** command is used to generate a block for the entire system. The syntax for the MMCS command is:

```
genfullblock <blockid> <machineSN> <psetsize>
```

The parameters for the command are:

- <blockid> The name of the entry created in the BGPBLOCK table.
- <machineSN> The serial number defined in the BGP MACHINE table (usually BGP).
- <psetsize> The ratio of compute nodes to one I/O node (depends on the hardware configuration).

The following command creates a block, named *System*, that uses every midplane on the system (see Figure 4-14):

```
mmcs$ genfullblock System BGP 8
```

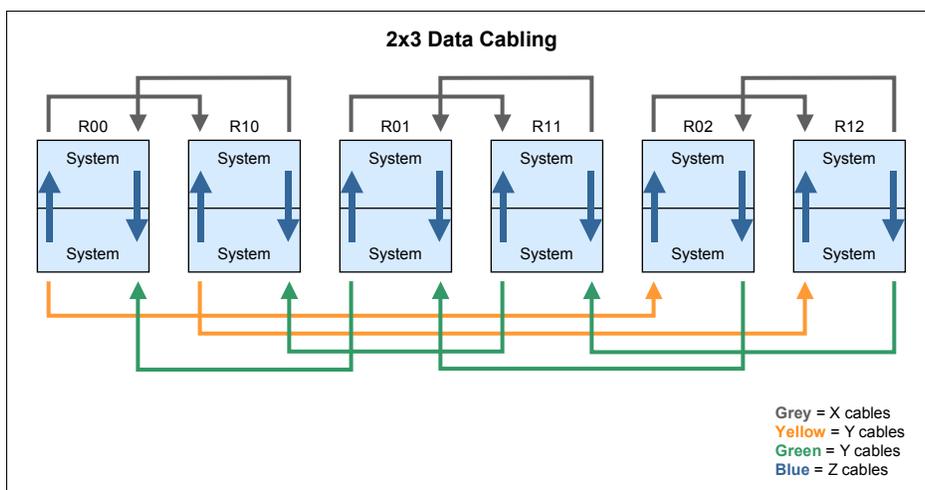


Figure 4-14 Block using all resources in the Blue Gene/P core



Midplane Management Control System

You use the Midplane Management Control System (MMCS) to configure and allocate blocks of compute nodes and I/O nodes and to run programs on the Blue Gene/P system. Referred to as the *high-level control system*, MMCS runs on the Service Node and provides an interface to the hardware level system components. You can access the MMCS directly using the MMCS console, Blue Gene Navigator, or through the various scheduler programs (using the scheduler APIs). In addition to the high-level control functions of the MMCS server, the following functions are incorporated into the server:

- ▶ MMCS acts as a relay, passing Reliability, Availability, and Serviceability (RAS) messages from the mcServer to the database.
- ▶ Environmental monitoring.

Evolution Note: Although much of the base MMCS from Blue Gene/L was maintained, there are differences in the Blue Gene/P version. One notable change is that MMCS no longer contains any low level, hardware debug type of functionality.

5.1 MMCS components

This section describes the components of MMCS and how they interact with the hardware and Blue Gene/P users. While most users use MMCS primarily through a job management subsystem, such as LoadLeveler, developers and system administrators want to use the cluster's resources more directly, through `mmcs_db` commands.

Figure 5-1 shows an overview of the conceptual framework on which the `mmcs_db` command supports and operates.

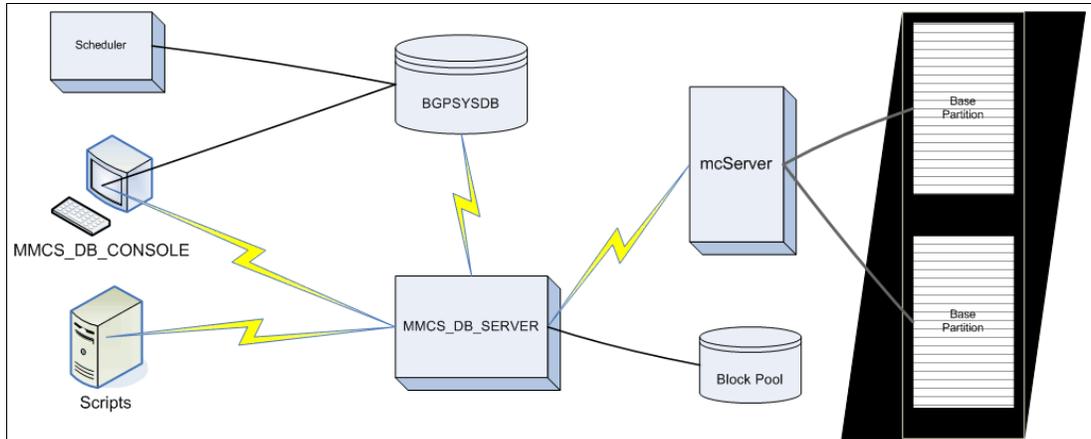


Figure 5-1 MMCS components

MMCS server handles requests from clients

The MMCS server runs on the Service Node and listens for incoming connections from MMCS clients or script clients. The MMCS server is capable of handling multiple client connections. It simultaneously performs other functions such as monitoring for RAS events, environmental monitoring and mailbox monitoring.

MMCS clients

MMCS commands are issued from either MMCS console or from a program or script through a socket connection. The `mmcs_db` commands can be sent to the MMCS server from any program that can communicate over a socket, including most scripting languages. MMCS console supports additional commands for querying or updating the central database. The job management subsystem interacts with `mmcs_db` through a set of programming interfaces to the database.

Blocks are defined in the database

MMCS console and script clients allocate only predefined blocks configured in the central database. System administrators define these blocks based on the needs of the users. In that respect, LoadLeveler has a greater capability. It can analyze current cluster resources and dynamically define blocks based on specific job requirements and cluster resource availability.

The MMCS server performs the actual block allocation, booting, switch configuration, and deallocation based on requests from its clients. It keeps track of allocated blocks within the Blue Gene/P cluster. These blocks are not freed automatically when a client disconnects. They remain in a pool of allocated blocks and can be subsequently selected and operated on by other clients. Because the blocks are allocated to a user by name, any client that is self-identified as that same user is eligible to select and use an allocated block. MMCS

console and LoadLeveler automatically set the current user name, but script clients must do this explicitly.

Note: The `setusername`, or `set_username`, MMCS commands are used to establish or change the identity of the current user.

mcServer communicates with the Blue Gene hardware

The clients are not aware of the existence of mcServer in the control system but it plays a key role in the process. Through mcServer the MMCS Server is able to monitor the availability of the hardware. mcServer also controls access to the hardware to avoid conflicts. See 3.2, “mcServer” on page 67 for more details.

5.2 Starting the MMCS server

The MMCS server, by default, is started by the BGPMaster process. BGPMaster monitors the control system processes and attempts to restart them if they would happen to end. See Chapter 3, “BGPMaster” on page 63 for more information. You can also start the MMCS server manually using the following command line:

```
/bgsys/local/etc/startmmcs
```

You can also use the `/bgsys/driver/ppcfloor/bin/mmcs_db_server` command.

The `startmmcs` command creates a log file using the syntax `<hostname>-mmcs_db_server-yyyy-mmdd-hh:mm:ss.log`. It also creates `<hostname>-bgdb0-mmcs_db_server-current.log` as a symlink to the log file. The link is updated to link to the new file each time the server is started. If the `mmcs_db_server` command is used to start the MMCS server, no log file is generated.

Important: Keep in mind that the MMCS server relies on the mcServer to maintain contact with the hardware. Thus, the mcServer must also be started manually if you are going to bypass BGPMaster. The `startmcserver` command (your local version) is also located in the `/bgsys/local/etc/` directory.

By default the MMCS server listens on port 32031 for incoming commands. The server only listens on the loopback interface (127.0.0.1), so only local clients can connect. The MMCS server communicates with the mcServer on port 1206.

5.3 Using MMCS console

This section details the starting and use of MMCS console. We cover some of the commonly used commands to demonstrate the use and output of the console.

5.3.1 Starting an MMCS console session

You use the `./mmcs_db_console` program to start a console session, and it is located in the `/bgsys/drivers/ppcfloor/bin` directory. By default there are no special authorities required to run the `./mmcs_db_console` program.

Important: Accessing the MMCS server is only allowed from the Service Node. Anyone who has access to the Service Node can connect to the MMCS server using the MMCS console.

MMCS console initializes by connecting to the database and attempting to connect to the MMCS server. It does not continue if it cannot obtain a database connection, but continues even if it cannot obtain an MMCS server connection. This permits access to the database while the server is not running.

5.3.2 Allocating blocks

Normally a user only uses a single console and a single block at a time. The user allocates the block using the **allocate** command, runs jobs using the **submitjob** command, and frees the block before ending the console using the **quit** command.

The **allocate** command establishes connections to the block after clearing any existing connections. It also reboots the block using the microloader, compute node, and I/O node images specified in the block definition in the database. Alternatively, you can use the **allocate_block** command which resets and establishes connections to the block but does not boot the block. You can then use the **boot_block** to boot a block allocated with **allocate_block**.

In some cases, a user might want to use the **allocate** command to allocate a block, use the **submitjob** command for a job, and then use **quit** while the job is running. To return and operate on the same block, the user starts MMCS server and uses **select_block** to reconnect to the previously allocated block.

There might be some situations in which a user wants two console sessions connected to the same block. For example, suppose a user started a **waitjob** command to wait for a program to end, and it does not appear to be ending. The console is hanging until the job ends, due to the **waitjob** command. The user could start a second console session, select the original block with **select_block**, examine the job status with **list bgpjob**, or stop the job by using the **kill_job** command.

5.3.3 Starting jobs

Running an application program with MMCS is done by using the database. The **submitjob** command adds a record to the BGPJOB database table and immediately returns. The MMCS server monitors the database. It starts jobs on initialized blocks by communicating with CIOD running on the block's I/O nodes. In this case, the CIOD is not aware of jobs that end and start within the block. The jobs are controlled by the master scheduler program.

When the job starts, the UID and GID are set to that of the user who submitted the job. Also the current directory is set to the one that is specified on the **submitjob** command. Any standard output is directed to text files in the directory specified on the **submitjob** command.

You can use the **wait_job** command to wait for the job to complete. You can also use the **list bgpjob** command to view the job's status, definition, and any error messages from job initialization.

5.4 MMCS commands

Table 5-1 contains a list of the supported MMCS commands for Blue Gene/P.

Table 5-1 Supported commands

Command	Syntax and description
!	! [shell command] This command escapes you to a subshell if no <command> is specified. Use "exit" to return to the mmcs\$ prompt. If the optional <command> is specified, it runs that command in a subshell and returns you to the mmcs\$ prompt. (Can only be used on the console)
#	# [<comment.>] This command indicates a comment line.
<	< <filename> This command pipes command input from <fileName> to MMCS server. (Can only be used on the console)
addbootoption	addbootoption <blockId ALL> <bootoption> addbootoption is used to add a boot option for a specific block or all blocks. This value will override entries in the db.properties file. Multiple options can be separated by commas, no spaces.
allocate	allocate <blockId> [htc=<smplduallvnm>] The allocate command performs select_block, allocate_block, boot_block and wait_boot for the specified block (block must be free). You may also use this command to allocate a block for HTC jobs to run on.
allocate_block	allocate_block <blockId> [options] allocate_block marks the block as allocated but does not boot it. Options include: <ul style="list-style-type: none"> ▶ no_connect - do not connect to block hardware ▶ pgood - reset pgood on block hardware ▶ diags - enable block to be created when components are in Service state ▶ shared - enable nodecard resources to be shared between blocks. Implies no_connect. ▶ svchost_options=<svc_host_configuration_file>
assignjobname	assignjobname <jobId> <jobName> Assigns a character job name to a specific jobId. (Can only be used on the console)
associatejob	associatejob <jobId> <blockId> Associates a previously created job with a pre-allocated block. (Can only be used on the console)

Command	Syntax and description
boot_block	<p>boot_block [update] [<options>] Initialize, load and start the block's resources.</p> <p>Options include:</p> <ul style="list-style-type: none"> ▶ uloader = <path> - microloader ▶ ioload = <path> - I/O node elf images. A comma separated list. ▶ cnload = <path> - compute node elf images. A comma separated list. ▶ virtual_node_mode - prepare the nodes to run the application on all cores ▶ no_global_interrupts - do not use global interrupts ▶ standalone - disable CIOD interface ▶ boot options - any other options are passed to the machineController on the boot command <p>Scenarios:</p> <ul style="list-style-type: none"> ▶ If no options are specified, the uloader, ioload, cnload, and boot options specified in the block definition are used ▶ If any options are specified, and 'update' is not specified, only the uloader, ioload, cnload, and boot options specified on the boot_block request are used ▶ If 'update' is specified, the uloader, ioload, cnload, and boot options from the boot_block command will be combined with the block definition. Any boot options will be added to those from the block definition. Any uloader, ioload, or cnload specified on the boot_block command will replace the specification from the block definition, but any uloader, ioload, or cnload not specified on the boot_block command will be taken from the block specification.
connect	<p>[<target>] connect [options] Connect to a set of a block's resources.</p> <p>Options include:</p> <ul style="list-style-type: none"> ▶ targetset=[temp perm] - create the mcServer target set as temporary (default) or permanent ▶ outfile=<filename> - direct the console messages to a file ▶ rasfile=<filename> - direct the ras messages to a file ▶ tee - direct the mailbox out to both the file and stdout ▶ no_ras_disconnect - do not disconnect the block on a fatal RAS event. ▶ pgood - reset pgood on block hardware
copyblock	<p>copyblock <existingblockId> <newblockId> Duplicate an existing block, along with all of its boot information</p>
createjob	<p>createjob [blockId] <executable> <outputDir> [mode] Defines a job in the BGPJOB table that will run <executable> in optional initialized <blockId>. createjob is used in conjunction with setjobargs and startjob. The <executable> parameter specifies the path to the program to be started. OutputDir specifies a writable directory to become the home directory for the job and to contain the output files for text written to stdout and stderr. The job will be in queued status and is not eligible to run until startjob is issued. (Can only be used on the console)</p>
debugjob	<p>debugjob <jobId> Starts the debugger on a running job</p>
delete	<p>delete <db2table> <id> Deletes the specified record id from the database. The only two valid tables for this command are BGPBLOCK and BGPJOB. (Can only be used on the console)</p>

Command	Syntax and description
deselect_block	deselect_block Stop working with the currently selected block.
disconnect	disconnect Disconnect from a set of block resources.
free	free <blockId> Releases the resources associated with the blockId. Marks the block free in the BGPBLOCK table. Before an HTC partition can be freed no jobs can be active on the partition. To end all running HTC jobs on a partition use the kill_htc_jobs command.
free_block	free_block Releases the resources associated with the currently selected block. Marks the block free in the BGPBLOCK table.
genBPblock	genBPblock <blockId> <machineSN> <psetsize> <cornerBP> <xsize> <ysize> <zsize> Generates a block for a set of base partitions (BP). The <psetsize> is the number of compute nodes for each I/O node (see Chapter 4, "Blocks" on page 79 for more explanation on creating blocks). For <cornerBP> specify the midplane by location, that is, Rxx-Mx of the first BP in the block. The size is provided in terms of number of midplanes in X, Y and Z dimensions. The <cornerBP> will be in the 0,0,0 position of the midplanes that make up the block. Except for the <cornerBP>, the midplanes included in the generated block depend on X,Y,Z cabling of your machine.
genblock	genblock <blockId> <midplane> <machineSN> <psetsize> Creates a block for the specified midplane.
genblocks	genblocks [blockIdprefix] <machineSN> <psetsize> Generates a block for each midplane in the system. If the blockprefix parameter is not supplied the block name will be the same as the position of the midplane (Rxx-Mx).
genfullblock	genfullblock <blockId> <machineSN> <psetsize> Generates a single block for the entire machine.
gensmallblock	gensmallblock <blockId> <midplane> <machineSN> <psetsize> <cnodes> <nodecard> Creates a sub-midplane block. The valid number of compute nodes (<cnodes>) is 16, 32, 64, 128, or 256.
getblockinfo	getblockinfo <blockId> Returns the boot information, boot mode and status of the specified block.
getjobinfo	getjobinfo <jobId>
getjobs	getjobs <jobs.xml> <blockId> Retrieves a list of jobs associated with the blockId identified in the command. The results are stored in the xml file.
help	help [command name] Prints the command summaries for all commands or, if named, a specific command.

Command	Syntax and description
job_trace	job_trace [blockId] on off Enables tracing of communications between mmcs and CIOD. Job trace might be enabled globally or for a specific block. If no block is specified, it is enabled globally and all new jobs will have tracing enabled regardless of how they have been set by previous invocations of job_trace. [blockId] identifies a block in the BGPBLOCK table. Job tracing is not enabled on current jobs. The next new job for a trace enabled block will be traced. Trace output is dumped to the mmcs log.
kill_htc_jobs	kill_htc_jobs [block=<blockId>] [user=<username>] [timeout=<seconds>] The kill_htc_jobs command kills all High Throughput Computing (HTC) jobs running by the supplied user and/or block. You must specify block or username if no block is selected. If a block is currently selected, all jobs on it are killed matching the optional username argument.
kill_job	kill_job <jobId> [timeout] Kill the specified job on the currently selected block. The optional timeout parameter overrides default specified by mmcs server. The command returns before the job is killed. You can use wait_job to wait for the job to be terminated.
killjob	killjob <blockId> <jobId> Kills the job on the specified blockId. The command returns before the job is killed.
list	list <db2table> [<value>] Returns the contents of the database table listed. The following are the only valid table/field combinations: <ul style="list-style-type: none"> ▶ bgpnode / location ▶ bgpnodecard / location ▶ bgpmidplane / location ▶ bgpmachine / serialnumber ▶ bgpblock / blockid (wild card with %) ▶ bgpproducttype / productid ▶ bgpjob / jobid ▶ bgpjob_history / jobid ▶ bgpeventlog / block (Can only be used on the console)
list_blocks	list_blocks Returns all of the currently allocated blocks. The output includes the user, number of consoles started and if the output is redirected to the console.
list_bps	list_bps Returns base partition information for the entire machine, including blocks that are booted on each of the midplanes.
list_htc_jobs	list_htc_jobs [<blockID> *ALL[<username>]] The list_htc_jobs command prints jobId, status, username, blockid, location, and executable for all active High Throughput Computing (HTC) jobs. Specify *ALL for <blockid> to see active jobs on any block matching an optional <username>. If both <blockid> and <username> are omitted, all active HTC jobs are returned.
list_jobs	list_jobs [<username>] Prints jobId, status, username, blockId and executable for all jobs that are in BGPJOB table, or those jobs in BGPJOB with optional <username>. Jobs that are running in HTC mode are also noted.

Command	Syntax and description
list_users	list_users Lists all mmcs users. The output returned includes thread number, block ID and if the output is redirected to the console.
locate	[<target>] locate [neighbors] [ras_format] Lists physical location (midplane, board, card, slot) of all nodes in allocated block. If target node is specified, lists the location of the specified node. Add [neighbors] to get the + and - neighbors in x,y,z dimensions for the specified node. Add [ras_format] to get location information in the format found in RAS events.
locaterank	locaterank <jobld> <mpirank> Locate a node using the jobld of a job and the MPI rank. Finds the job <jobld> in the job table or job history table, and using the attributes of the job, such as the block and the MPI mapping options, displays the physical location of the node with MPI rank <mpirank>.
quit	quit Exit from the MMCS console session. (Can only be used on the console)
reboot_nodes	[<target>] reboot_nodes [<blockid>] [<timeout>] Reboot compute nodes or specific I/O nodes. The blockid and timeout parameters are optional. If used blockid must precede timeout.
redirect	redirect <blockld> on off Redirect I/O node output for the specified block to the mmcs console. Directs subsequent mailbox output back to the socket connection that this command is received on. Allocating or freeing the block will stop the mailbox redirection.
replyformat	replyformat [0 1] With no parameters replyformat displays the current reply format, 0 or 1. 0 means that each reply is returned as a single text line, embedded newlines are indicated by a semicolon and then end of the reply is indicated by a new line 1 means that each reply is returned as one or more text lines, each line terminated by a newline. The end of the reply is indicated by a ' replyformat OK 1.
select_block	select_block <blockld> Select an already allocated block to work with. select_block does not initialize the block nor reset connections. This command is intended to be used to reconnect to an allocated block from a another mmcs_db console session.
set_username	set_username <username> Set username for block allocation and job scheduling. You cannot issue commands against a block that is allocated to another user. When the MMCS console starts, this command is automatically issued on behalf of the current user, and so does not need to be issued by the MMCS console user unless they want to run under a different user ID. However, it must be issued as the first command by any mmcs_db script.
setblockinfo	setblockinfo <blockld> <mloader> <cnload> <ioload> [<bootoptions>] Sets the boot images for a block. To set multiple loads for cnload or ioload, use multiple paths, separated by commas, with no spaces.
setbootoptions	setbootoptions <blockld ALL> <bootoptions NULL> Sets boot options for a block (blockld) or globally (ALL). To set multiple options separate by commas (no spaces).

Command	Syntax and description
setdebuginfo	setdebuginfo <jobld> <debugger> [<args>] Set the debugger image and optional debugger arguments for a specific job.
setjobargs	setjobargs <jobld> <args...> <envs....> Set arguments and environment variables for a specific job. Args is a list of arguments passed to the program. Envs is a list of environment variable settings of the form name=value. The environment variables will be accessible to the program when it is running on a compute node. (Can only be used on the console)
setusername	setusername <username> Set username for block allocation and job scheduling. You cannot issue commands against a block that is allocated to another user. When the MMCS console starts, this command is automatically issued on behalf of the current user, and so does not need to be issued by the MMCS console user unless they want to run under a different user ID. However, it must be issued as the first command by any mmcs_db script.
sleep	sleep <seconds>
sql	sql <sql-string> OR execute <sql-string> This command can be used to run an sql statement from the MMCS console. It cannot be used to run a statement that returns a value (i.e., queries)
starthwpolling	starthwpolling [ncseconds [scseconds [lcseconds]]] Start polling the hardware for environmental readings. See 5.7, "Environmental Monitor" on page 104
startjob	startjob <jobld> Schedules the specified job <jobld> to be started. Marks the job as eligible to start in the BGPJOB table. The job will not be started until the block is available and no prior jobs for the block are eligible to start. (Can only be used on the console)
status	[<target>] status Lists nodes in allocated block, the type of each node, and whether it is running a program or vacant. If the block has been booted, it will show that a program is running.
stophwpolling	stophwpolling Stop polling the hardware for environmental readings. See 5.7, "Environmental Monitor" on page 104
submit_job	submit_job <executable> <outdir> [mode] [args] [envs] Starts an <executable> running on the currently selected block. <ul style="list-style-type: none"> ▶ <executable> parameter specifies the path to the program to be started. ▶ <outDir> specifies a directory to become the home directory for the job and to contain the output files for text written to stdout and stderr. The bgpadmin user MUST have permission to access this directory either because it is world executable or because the bgpadmin group owns it and it is group executable. ▶ [mode] is optional. The possible modes are SMP, DUAL or VN. If omitted, the job will run in SMP mode. ▶ <args> is a list of arguments passed to the program. ▶ <envs> is a list of environment variable settings of the form name=value. The environment variables will be accessible to the program when it is running on a compute node. The response to this command contains a job number which can be used with waitjob or killjob.

Command	Syntax and description
submitjob	submitjob <blockId> <executable> <outdir> [mode] [args] [envs] Create a job in the DB to run <executable> in <blockId> and schedule it to be started. This is the same command as submit_job, but with an additional BlockId parameter. This form of the command causes the specified block to be selected before command is executed. See submit_job for full description of parameters.
username	username Returns the current username.
version	version Displays the current MMCS version.
wait_boot	wait_boot [<minutes>] Wait for the selected block to complete booting before returning control. <minutes> specifies the maximum time to wait, in minutes. The default is 15 minutes. The command does not complete until the block is fully initialized to the point that mmcs can communicate with CIOD on all of the I/O nodes or the wait time has expired
wait_job	wait_job <jobId> Waits for the job with the specified jobId to end on the selected block. Returns immediately if the job is no longer running or failed to start.
waitjob	waitjob <blockId> <jobId> Waits for the job with the specified jobId to end on the specified block. Returns immediately if the job is no longer running or failed to start.
write_con	[<target>] write_con <console-command> Send <console-command> to target node for execution. Output will be returned to mailbox (either console or I/O node log).

Table 5-2 contains commands that accept an optional <target>, which can be used to specify the nodes on which to apply the operation.

Table 5-2 Node commands

Target expression	Meaning
{*}	all nodes (default)
{i}	all I/O nodes
{c}	all compute nodes
{nc}	all node cards
{lc}	all link cards
{l}	all link chips
<N>	single node
<N>,<N>...	list of nodes
<N>-<N>	subrange of nodes
<N>-<N>,<N>,...	combination of range and list
Individual cores can be specified by appending a period followed by the core number to the target	
{1}.0	core 0 on node 1

5.5 Output

The block control functions run as threads under the MMCS server process on the Service Node. Therefore, any output that is directed to standard output or standard error is directed to the MMCS server process console.

5.5.1 Command and response

The response to an `mmcs_db` command consists of the word `OK` followed by optional messages or the word `FAIL` followed by an error message (see Figure 5-2). The internal representation of the response string depends on the reply format in effect at the time.

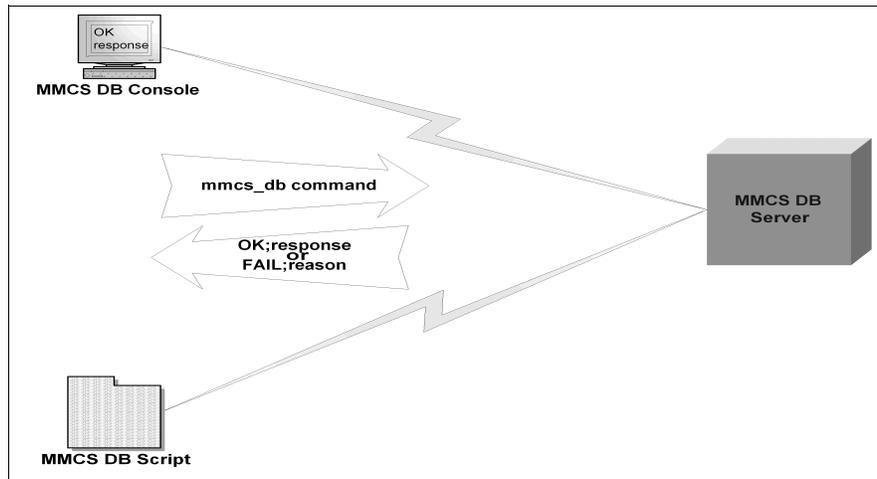


Figure 5-2 Command and response (format 0)

The `replyformat` command

The `replyformat` command is used to set the response received from the MMCS server. The command takes a single, optional argument:

```
replyformat 0      establishes the single-line reply mode (the old format).
replyformat 1      establishes the multi-line reply mode (the new format).
replyformat        (no parameters) displays the current reply mode.
```

The old format has a 16 K length limit on all replies. If the limit is exceeded, the reply is truncated with the message:

```
;ABORT;reply length exceeded
```

MMCS server defaults to the newer multi-line format. However, scripts default to the old single-line format to maintain compatibility.

replyformat 0

This response consists of a single string, terminated by a single newline character at the end of the string. The first word in the string is either `OK`, indicating success, or `FAIL`, indicating failure. If additional data follows, `OK` or `FAIL` is followed by a semicolon and the additional data. For example, the response to a successful `submitjob` command is:

```
OK;jobId=xx\n
```

Multiline responses have embedded new line characters that are replaced with semicolons. Each semicolon represents a new line character that was replaced prior to being sent over the socket connection from `mmcs_db` to the client. The MMCS server replaces these

semicolons with new line characters before writing them to the console window, so a multiline response looks the way it does if it is printed from the local process. However, it is the script's responsibility to parse the response and (optionally) replace the semicolons with new line characters.

replyformat 1

The response content is the same as with reply format 0. Only the format differs. The response consists of OK or FAIL in the first line, optionally followed by additional data. The response is sent as multiple newline-terminated strings. There is no limit to the number of strings that can be sent in the reply.

The last line is followed by a line consisting of the null character followed by a newline (`\0\n`).

5.5.2 Mailbox output

Each node on an allocated block has a mailbox associated with it and the block's thread in MMCS server monitors the mailbox for output. Output from programs running on the compute nodes and I/O nodes can be written to the mailbox and processed by the mailbox thread.

Mailbox output is the I/O Node output. This is primarily debug and command output from I/O node kernel and compute node kernel, because application program output is handled separately.

The MMCS server has various options for directing this mailbox output:

- ▶ **Default:** If you do nothing, the mailbox output is directed to the MMCS console. If you have redirected the MMCS server standard output to a file, you can view the mailbox output in that file. However, the MMCS server output contains data from potentially many mailboxes.
- ▶ **Outfile parameter:** You can specify `outfile=filename` on the **allocate** and **allocate_block** commands to have the mailbox output directed to a file.
- ▶ **Redirection:** The MMCS server has a **redirect** command that sends the mailbox output back over the socket connection.

5.5.3 Job output

In MMCS the **submitjob** command is used to start an application program. It causes the program's standard output to be directed to a file. This file is created in the directory specified on the `submitjob` command. It is named with the block name, job number and output stream, such as `R00-M0-26.stdout`.

5.5.4 Redirection

The **redirect** command is an exception to the response format rule given previously. The **redirect** command causes the MMCS server mailbox thread to send all of its output back over the same socket connection. The mailbox data is sent over the connection as it is received from the mailbox. The client issuing the **redirect** command should immediately start looping and reading the `mmcs_db` socket connection. Semicolons should not be replaced with new line characters. The socket connection is reset when the mailbox thread is terminated by an **allocate**, **allocate_block**, or **free** command.

Note: The **redirect** command is not available in the Navigator's version of the MMCS console. After a block is allocated, or **select_block** is used, a button displays that allows you to view the selected block's output.

5.6 CIOD

Evolution Note: In Blue Gene/P, the MMCS server monitors both blocks and jobs. These were separate processes in Blue Gene/L. In the MMCS server was responsible for block functions and a process called CIODB was responsible for monitoring the database for job requests and communicating with the I/O nodes.

CIOD is a user-mode daemon that runs on the I/O node and provides a bridge between the compute nodes and the outside world. CIOD takes messages from multiple input sources, processes the message, and returns a result. CIOD accepts messages from three sources:

- ▶ The control system sends messages using the CioStream and DataStream protocols on two socket connections from the service node. The control system sends messages to load and start a new job, send standard input, and send a signal to a running job. CIOD sends standard output, standard error, and messages to report status.
- ▶ Compute nodes send messages using the CIO protocol on virtual channel 0 of the tree device. The messages are for both control, function shipping I/O, and debugging.
- ▶ An external debug server sends messages using the CioDebug protocol on pipes from another I/O node process.

For more information about CIOD on the I/O nodes, see Chapter 11, “Configuring the I/O nodes” on page 161.

5.7 Environmental Monitor

The Environmental Monitor is another facet of MMCS. Monitoring on Blue Gene/P consists of collecting data from the running hardware, such as temperatures, clock frequency, fan speeds and voltages. Generally referred to as environmentals, this information can be useful in indicating system health or debugging problems. Environmentals are automatically collected every five minutes by default. The values collected are stored in the database for three months. Current and historical data can be viewed in Blue Gene Navigator by clicking **Environmental Queries** as shown in Figure 5-3. To assist in the efficient administration of the system, the Environment Monitor will also generate RAS events when abnormal readings are encountered.

The screenshot shows the Blue Gene Navigator interface. At the top, it says "Blue Gene Navigator" and "Welcome, bgpuser". On the left is a navigation menu with categories like Administration, Resources, and Environmentals. The main content area has a status bar with "Attention" and a warning icon. Below that are tabs for "Environmental Queries" and sub-tabs for "Bulk Power Modules", "Link Card Power", "Node Card Power", and "Service Card Power". A table shows the following data:

Location	Time	Ref Input Fault	PLL Lock Fault	Frequency
R00-K	8/29/07 8:35:28 AM	False	False	850.000
R01-K	8/29/07 8:35:28 AM	False	False	850.000

At the bottom of the table, there are navigation buttons: "First", "Previous", "Next", "Page 1", "Page size: 50", and "Apply".

Figure 5-3 Environmental Queries

How it works

The Environmental Monitor starts when the MMCS server is started. The MMCS server starts the envMonitor thread. In turn, the envMonitor thread starts individual threads to monitor each of the card types and a separate thread to purge stale data from the database. The threads that monitor hardware read the environmental data from mcServer and the Machine Controller (MC) and write the results to the database. By default each of these threads sleep for five minutes. The thread responsible for purging data is configured to sleep for thirty seconds at which time it wakes up and checks for a kill request from the MMCS server. The types of cards monitored are Service cards, Link cards and Node cards. The data for the Bulk Power Modules, Fan Modules and Clock cards are collected from the Service card.

Note: The lower midplane (midplane 0) is considered to be the master. The data for the Bulk Power Modules and the Clock Card is collected by the Service Card in the bottom midplane.

The reason multiple threads are used allows flexibility with the polling intervals for different types of hardware. The default values can be altered by adding (or changing) entries in the db.properties file. There are situations that might arise when you would want to temporarily disable the monitoring functions. As a general rule before changing or disabling the monitor you should check with IBM support.

Attention: The range of values that can be used to override the default polling interval is between 60 and 1800 seconds.

One instance that you would need to briefly stop polling is if you would need to reset the polling intervals for a specific type of hardware. The monitoring can be stopped by issuing the **stophpolling** command from the MMCS console without impacting users on the system. At this point there are two possible ways to alter the polling intervals. The first method would be to update the db.properties file with one or more of the following:

Service Cards	mmcs_envs_sc_interval_seconds=xxx
Link Cards	mmcs_envs_lc_interval_seconds=xxx
Node Cards	mmcs_envs_nc_interval_seconds=xxx

As mentioned earlier, the default is hard coded into the monitor code with a value of 300 seconds. The first time you change the values you will have to add the variables to the db.properties file. After the values are updated the monitor can be restarted, again through MMCS console, with the **starthpolling** command. Using this method overrides the default settings each time the monitor is started.

The second approach is to change the values for the current session of the Environmental Monitor. That is to say that, as long as the monitor is not stopped, the values supplied by this method will be used. We will use the same two commands that were used in the first example (updating db.properties). The process begins with stopping the monitor using the **stophpolling** command at an MMCS prompt. When you restart the polling with the **starthpolling** command you can supply up to three numeric values to control the rate at which the monitor polls. The parameters are ncseconds, scseconds, and lcseconds, which represent the number of seconds to wait before polling the Node Card, Service Card and Link Card, respectively. The following example of the **starthpolling** command would cause the Node Card data to be updated every 60 seconds, the Service Card every 600 seconds and the Link Cards once every 120 seconds:

```
mmcs$ starthpolling 60 600 120
```

The parameters are dependent on their position in the command so you need to provide a value in each of the positions.

Blue Gene/P environmental database tables

Under normal circumstances the Navigator is the means of viewing environmental data on the Blue Gene/P hardware. Because the data is stored in the hardware database you can also access it directly by running queries on the following tables:

- ▶ BGPBULKPOWERENVIRONMENT
- ▶ BGPCLOCKCARDENVIRONMENT
- ▶ BGPFANENVIRONMENT
- ▶ BGPLINKCARDENVIRONMENT
- ▶ BGPLINKCARDPOWERENVIRONMENT
- ▶ BGPLINKCHIPENVIRONMENT
- ▶ BGPNODECARDENVIRONMENT
- ▶ BGPNODECARDPOWERENVIRONMENT
- ▶ BGPNODEENVIRONMENT
- ▶ BGPSERVICECARDENVIRONMENT
- ▶ BGPSRVCCARDPOWERENVIRONMENT

The tables associated with the Environmental Monitor use both location and time as the primary key. See Appendix B, "Blue Gene/P hardware naming convention" on page 191 for a complete description of locations.

RAS events that are generated by the Environmental Monitor are written to the BGPEVENTLOG table.

Note: Each of the rows in the BGPEVENTLOG have a time stamp. There might be a delay between when an error occurs and when the Environmental Monitor polls the hardware. The time that is entered into the database is the actual time of the event, not when the hardware was polled.

Table 5-3 lists the events that generate an RAS event in the BGPEVENTLOG table.

Table 5-3 RAS event triggers

Hardware	Event
Service Card	Card cannot be contacted
Bulk Power Modules	<ul style="list-style-type: none"> ▶ Module not contacted ▶ Module not responding ▶ General warning ▶ General fault ▶ Temperature warning ▶ Temperature fault ▶ Current warning ▶ Current fault ▶ Voltage fault ▶ Output 0 not enabled ▶ Output 0 fault
Fan Modules	<ul style="list-style-type: none"> ▶ Fan Module not contacted ▶ Fan not responding ▶ Temperature limit exceeded ▶ Fan failed ▶ Fan speed out of spec (Actual speed 10% above or below desired speed)

Hardware	Event
Node Cards	<ul style="list-style-type: none"> ▶ Card not contacted ▶ Over temperature
Link Cards	<ul style="list-style-type: none"> ▶ Card not contacted ▶ Over temperature

Resources required

Based on the number of tables and the type of data collected for the various pieces of hardware we can make a few general assumptions about the amount of storage required to support the Environmental Monitor. Table 5-4 shows an estimate of the amount of storage required using the default five minute polling period.

Table 5-4 Approximate storage requirements

Hardware	1 Day(per rack)	7 Days(per rack)	30 Days(per rack)	90 Days(per rack)
Link Cards and Link Chips	~700 Kb	~4.9 Mb	~21 Mb	~63 Mb
Node Cards and Nodes	~10 Mb	~70 Mb	~300 Mb	~900 Mb
Service Cards	520 Kb	~3.64 Mb	~15.6 Mb	~46.8 Mb
Total	~11.2 MB	~78.54 Mb	~336.6 Mb	~1 Gb

By default the data collected by the Environmental Monitor will be purged every three months. This value can be changed by updating the `mmcs_envs_purge_months` entry in the `db.properties` file.

Blue Gene Navigator

The environmental results, both current and historical, can be viewed through the Navigator interface. In Figure 5-3 on page 104 you can see that there are tabs for all of the hardware that we have discussed in this section. Each of the tabs reports a variety of data. For example, Figure 5-4, shows the results for the Link Cards tab. On the left of the panel is the location of the card followed by the time stamp of when the data was reported. In the next few columns you see the most recent temperature of the sensors on the card and, in most cases, information about the power. In Navigator the temperatures are always color coded with blue being the coolest and red representing the warmest temperatures.

Tip: The color coding in the Environmental Queries pages of the Navigator can be adjusted to suit your environment. In the `/bgsys/local/etc/navigator-config.xml` file, add the following Environmentals container (shown with default values):

```
<environmentals>
  <min-temp>25.0</min-temp>
  <max-temp>50.0</max-temp>
</environmentals>
```

Any temperatures that are below the `min-temp` display in blue. Temperatures above the `max-temp` show in red. Values between the two settings are shades of blue, green, and red. You can reset the values and save your changes. The new settings take effect the next time you start the Navigator server. These settings only affect the colors that display in the Navigator.



Figure 5-4 Environmental Queries, Link Card tab

Many times the amount of data returned can be overwhelming. Click **Filter Options** just below the hardware tabs to open the filter window. By selecting one or more of the various column heads you can refine the results returned by Environmental Queries. In Figure 5-5 the amount of data is reduced by selecting **Most recent measurement for each location** and only the Link cards that have a *Max ASIC Temp* over thirty degrees.

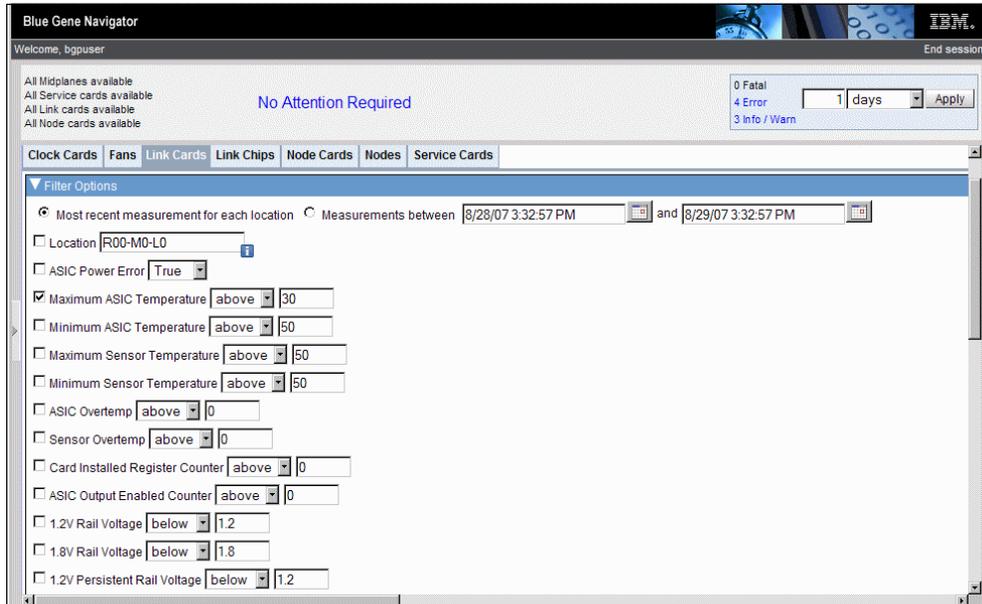


Figure 5-5 Environmental Queries Filter Options

As you can see in Figure 5-6, after clicking **Apply Filters**, the number of link cards returned by the query is only six.

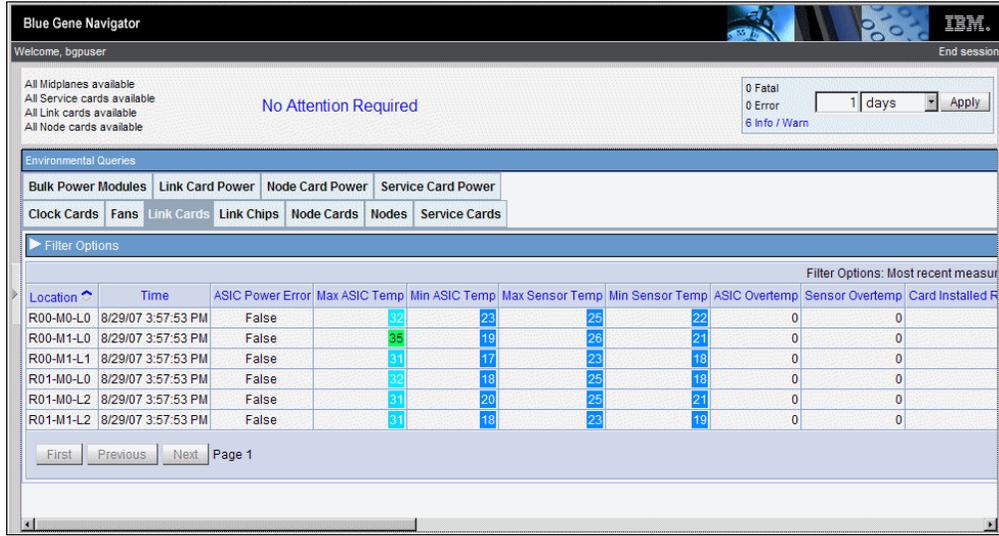


Figure 5-6 Filter results

Running your queries through the Navigator in the Environmental Monitor is much simpler than submitting long strings on the command line, and the results are far more readable.



High Throughput Computing

High Throughput Computing (HTC) first became available in Blue Gene/P with V1R2M0. Contrary to the traditional High Performance Computing (HPC) model, High Throughput Computing may use each of the compute nodes in a partition to run a different executable. HTC mode on Blue Gene/P is completely integrated into the Midplane Management Control System (MMCS), the control system of BG/P. Integration into MMCS means that the control system is fully aware of all HTC jobs. Those jobs are represented in the Blue Gene database when they are running and can be viewed from either the MMCS console on the service node or by using the Blue Gene Navigator from a browser. When jobs complete, information about the job is stored in the job history database table.

6.1 How it works

In the Blue Gene/P version of HTC, jobs enter the system via submit clients that are running on a Submit Node. The Submit Node will usually be a Front End Node but could also be the Service Node. The submit client connects to a software multiplexer (mux) that is running on the same node. The mux is then responsible for forwarding communications from each of its submit clients to the Service Node's submit server daemon.

The primary role of the submit server daemon is to manage system resources. When a partition boots in HTC mode the available partition resource states are managed in the submit server daemon's process memory so that a fast lookup can be done. Once the submit server daemon accepts and authenticates the incoming connection the submit server daemon determines if the necessary resources are available to complete the submit client's request. Based on the availability the request is accepted or denied. If the request is accepted the submit server daemon returns the allocated resource back to the submit client.

Figure 6-1 displays an overview of the High Throughput Computing (HTC) architecture.

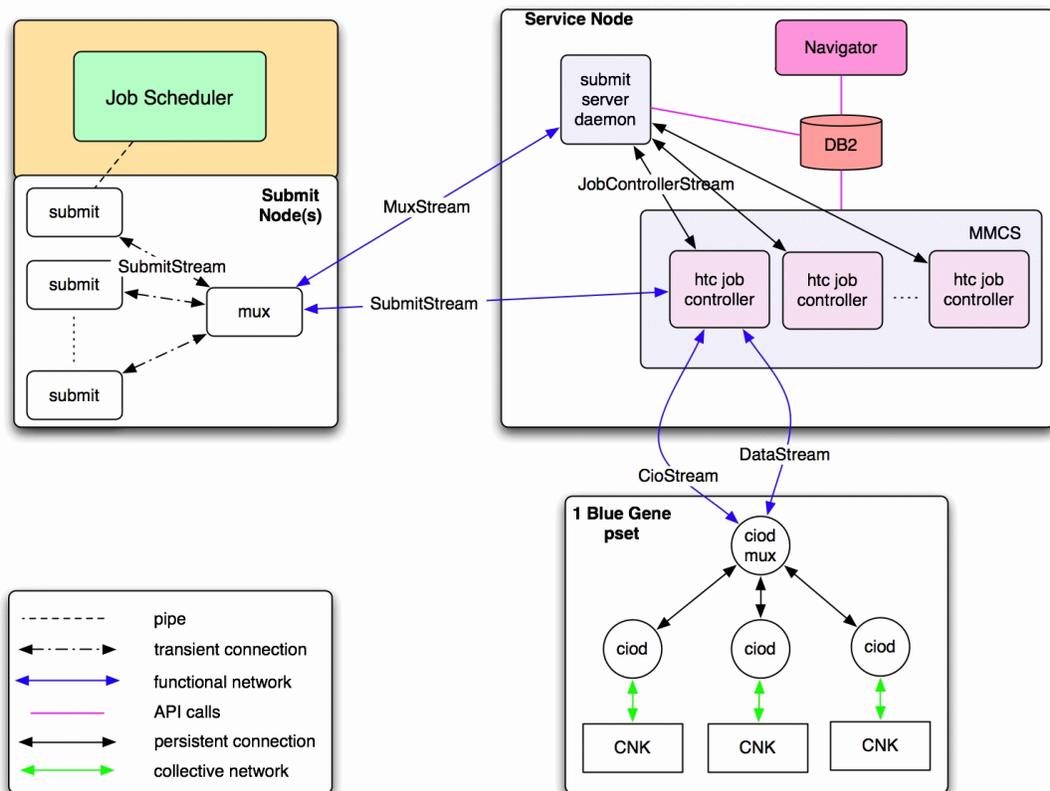


Figure 6-1 HTC Overview

For each HTC partition that is booted there is an associated HTC job controller started by the MMCS Server. The job controller connects to the submit server daemon and reports its availability and a list of managed resources. In return, the submit server daemon sends a list of active submit muxes and details on how to connect to them. The connection between the job controller and the submit server daemon is made on a configurable port and remains open as long as the partition is allocated.

The job controller also opens a connection to each Control I/O Daemon (CIOD) running in the partition, or the CIOD mux that represents the collection of all the CIODs running on the I/O

node. These connections are used to handle both job control messages (start/stop/etc.) and job data (stdin/stdout/stderr).

Once the submit client receives an allocated resource back from the submit server daemon it then sends the start (run) job request to the HTC Job Controller that owns the compute resource. All submit requests flow through a mux that has a persistent connection to the submit server daemon and every HTC job controller. Outside of allocating compute resources, which is done via the submit server daemon, all job management requests (starting/stopping/etc) flow between the mux and the HTC Job Controller. Additionally all data flow (stdin/stdout/stderr) travels between the mux and the HTC Job Controller.

6.2 Security

In this section we will examine the security features in the HTC design. There are three areas that we will focus on, submit client to submit mux, submit mux to the submit server and the submit mux to the job controller.

6.2.1 Submit client to submit mux

The submit mux listens on *localhost* only for connections from submit clients. There is no authentication between the mux and submit clients since it is expected that anyone with a login to a Front End Node is automatically authorized to use the submit command. Installations that want to prevent specific users or groups from accessing the submit command can do so by using the standard Linux file access control mechanisms.

6.2.2 Submit mux to submit server authentication

There is a challenge/response authentication protocol for submit muxes attempting to connect to the submit server daemon running on a Service Node. This is modeled after the mpirun to mpirun daemon method of authentication done on Blue Gene/P. Once the mux has authenticated to the submit server daemon it becomes a trusted service provider for submit clients.

The challenge password that the submit mux uses on the Submit Node is stored in the `/bgsys/local/etc/submit_mux.cfg` file. The owner of the file is `bgsadmin` and the group is `bgsadmin`. Both the owner and group have RW access while all others have no access. The challenge password that the submit server daemon uses on the Service Node is stored in the `/bgsys/local/etc/submit_server.cfg` file. The owner of the file is `bgsadmin` and the group is `bgsadmin`. Both the owner and group have RW access while all others have no access. The challenge password has to match in both `/bgsys/local/etc/submit_mux.cfg` and `/bgsys/local/etc/submit_server.cfg` in order for the submit mux to get authenticated to the submit server daemon.

6.2.3 Submit mux to HTC job controller

A HTC job controller connects to each submit mux after registering itself with the submit server daemon. Authentication is done according to the following procedure:

1. The submit mux generates a random string and sends this value to the submit server daemon as part of the registration process.
2. The submit server daemon sends this random string to each registering job controller.
3. The HTC job controller sends this random string to the submit mux when it connects.

4. The submit mux verifies that the random string sent by the HTC job controller matches the string generated in step 1.

6.3 Setting up HTC

There is some required setup to enable HTC mode. This section will walk you through each of necessary steps.

6.3.1 startsubmitserver (on the Service Node)

A template of the startsubmitserver file is provided in the `/bgsys/drivers/ppcfloor/bin` directory. Logged in as `bgpadmin`, copy this file to the local configuration using the following command:

```
cp /bgsys/drivers/ppcfloor/bin/startsubmitserver.tpl
/bgsys/local/etc/startsubmitserver
```

After the file has been copied verify that both the owner and group are set to `bgpadmin`. The authority to the file should be `RWX` for both the owner and group, public should have no authority to access the file. Ownership and authorities can be set with the following commands:

```
chown bgpadmin:bgpadmin /bgsys/local/etc/startsubmitserver
```

```
chmod 770 /bgsys/local/etc/startsubmitserver
```

6.3.2 bgpmaster.init

Starting with V1R2M0 the template file, `/bgsys/drivers/ppcfloor/bin/bgpmaster.init.tpl`, was updated to start the submit server daemon. Example 6-1 shows the template file.

Example 6-1 V1R2M0 and later bgpmaster.init file

```
# The servers are initially started in the order that they are defined
#
define    mcserver          startmcserver
define    mmcs_server      startmmcs
define    mpirund          startmpirund
define    navigator_server  startnavigator
define    realtime_server  startrealtime
define    submit_server    startsubmitserver

start    mcserver
start    mmcs_server
start    mpirund
start    navigator_server
# start    realtime_server
# Uncomment start realtime if you or your scheduler are using the
# real-time bridge APIs and have configured the database schema
# for real-time.
start    submit_server
```

These changes need to be applied to the production `bgpmaster.init` file. If you haven't made any local configuration changes to the `/bgsys/local/etc/bgpmaster.init` file the template file can just be copied to your local configuration using the following command:

```
cp /bgsys/drivers/ppcfloor/bin/bgpmaster.init.tpl /bgsys/local/etc/bgpmaster.init
```

If you have made changes to the local version of the init file you can just edit the `/bgsys/local/etc/bgpmaster.init` file and add the lines pertaining to `submit_server`.

The authorities on the `bgpmaster.init` file should be 644, with both the owner and group set to `bgpadmin`.

6.3.3 Submit mux on the submit nodes

The `submit_mux.cfg` file contains configuration settings used by init scripts to automatically start the submit mux on Submit Nodes. Typically a Submit Node is a Front End Node (FEN) where the `submit` command is executed. The configuration file also contains the challenge password used to authenticate a submit mux with the submit server daemon running on the Service Node. The `submit_mux.cfg` file needs to be created in the `/etc` directory with the proper file authorities and configuration values set.

The template for `submit_mux.cfg` is located at `/bgsys/drivers/ppcfloor/bin/submit_mux.cfg.tpl`. This file can be copied by the Blue Gene System Administrator using the following command:

```
cp /bgsys/drivers/ppcfloor/bin/submit_mux.cfg.tpl /bgsys/local/etc/submit_mux.cfg
```

After the file is copied the correct file authorities need to be applied. The owner of the file should be `bgpadmin` with RW authority, group authority should be `bgpadmin` with RW authority and all others should have no access to the file. The correct file authority settings is very important since the challenge password is contained in the file. The following commands set the ownership and privileges on the file:

```
chown bgpadmin:bgpadmin /bgsys/local/etc/submit_mux.cfg
```

```
chmod 660 /bgsys/local/etc/submit_mux.cfg
```

Customizing the `/bgsys/local/etc/submit_mux.cfg` configuration file

Typically you will only have to change two settings in the `submit_mux.cfg` file, the host name of the Service Node and challenge password.

- ▶ Set `SUBMIT_SERVER_HOSTNAME` to your Submit Server (Service Node) host name, e.g., "bgp.yourSite.com"
- ▶ Set `CHALLENGE_PASSWORD` to the challenge password that is configured in the `/bgsys/local/etc/submit_server.cfg` file on the Service Node where the Submit Server is running. The values for `CHALLENGE_PASSWORD` in `/bgsys/local/etc/submit_mux.cfg` (file on Submit Node/FEN) and `/bgsys/local/etc/submit_server.cfg` (file on Service Node) must match.

Starting the submit mux daemon on the Submit Node

Copy the init script to `/etc/init.d`, install the submit mux daemon to start automatically at the default run levels, and start it manually. The template for the `ibm.com-submit_mux` init script is located at `/bgsys/drivers/ppcfloor/bin/ibm.com-submit_mux.tpl`. This file can be copied by the Blue Gene System Administrator using the following command:

```
cp /bgsys/drivers/ppcfloor/bin/ibm.com-submit_mux.tpl
/etc/init.d/ibm.com-submit_mux
```

After the file is copied the correct file authorities need to be applied. The owner of the file should be *bgpadmin* with RWX authority, group authority should be *bgpadmin* with RWX authority and all others should have no access to the file.

```
chown bgpadmin:bgpadmin /etc/init.d/ibm.com-submit_mux
```

```
chmod 770 /etc/init.d/ibm.com-submit_mux
```

To install the submit mux daemon issue the following command:

```
/usr/lib/lsb/install_initd -v ibm.com-submit_mux
```

Then start the daemon using this command:

```
/etc/init.d/ibm.com-submit_mux start
```

6.4 MMCS Console commands

The MMCS console provides the capability to boot blocks of compute nodes and I/O nodes. HTC jobs can't be started from the console (this must be done using the submit interface instead), but administrator commands to list and terminate running HTC jobs are available. Many of the MMCS commands that are applicable to HTC or HPC. Below is a list of commands that can be used for either HTC or HPC.

Block commands

- ▶ addbootoption
- ▶ allocate
- ▶ allocate_block
- ▶ boot_block
- ▶ copyblock
- ▶ deselect_block
- ▶ free
- ▶ list_blocks
- ▶ select_block
- ▶ setblockinfo
- ▶ setbootoptions
- ▶ wait_boot

Job commands

- ▶ getjobinfo
- ▶ getjobs
- ▶ job_trace
- ▶ kill_job
- ▶ killjob
- ▶ wait_job
- ▶ waitjob
- ▶ list_jobs

For a complete list of MMCS Console commands see Table 5-1 on page 95.



RAS Messages

The RAS (Reliability, Availability and Serviceability) reporting structure has changed considerably from Blue Gene/L. Rather than a text based format Blue Gene/P uses a message id format that provides a consistent interface to analyze messages.

7.1 Blue Gene/P RAS

RAS messages have evolved from the text based messages used in Blue Gene/L to messages that have a defined structure in Blue Gene/P. Basic information such as a message ID, severity, location and text related to the event are available at first glance on the *RAS Event Log* page in Navigator. Additional information such as the component, raw data, and service suggestions can be obtained by clicking the additional information icon next to the message. Formatting the messages in this manner improves the usability by providing uniformity in the delivery of RAS events.

The Blue Gene/P version of Diagnostics also uses the RAS infrastructure to report errors and progress. This feature provides a central location that administrators and support can begin looking for events.

Blue Gene/P also provides the ability for users to define RAS events from within their program. There are forty precreated, user definable RAS events that can be used to log messages in the event log.

You can preview all of the RAS messages with the basic information included by clicking on the *RAS Message Types* link in the Navigator. This information can also be found in the file `/bgsys/drivers/ppcfloor/doc/RasEventBook.htm` which can be viewed with a browser.

7.2 RAS event message flow

RAS events can be generated from several sources:

- ▶ The kernel can create an event and send it to the Machine Controller (MC) using the mailbox protocol.
- ▶ The card controllers in the MC can also create RAS events.
- ▶ Diagnostics generate RAS events to report test results.

Events coming from the kernel or hardware are read and interpreted by the Machine Controller and passed to a mcServer listener. Messages created by Diagnostics are sent directly to the mcServer listener. The mcServer relays the events to a registered listener on the MMCS server. The MMCS server then logs the RAS event to the database at which time it can be viewed from the Navigator or by querying the database directly. Figure 7-1 shows the path that the RAS messages follow.

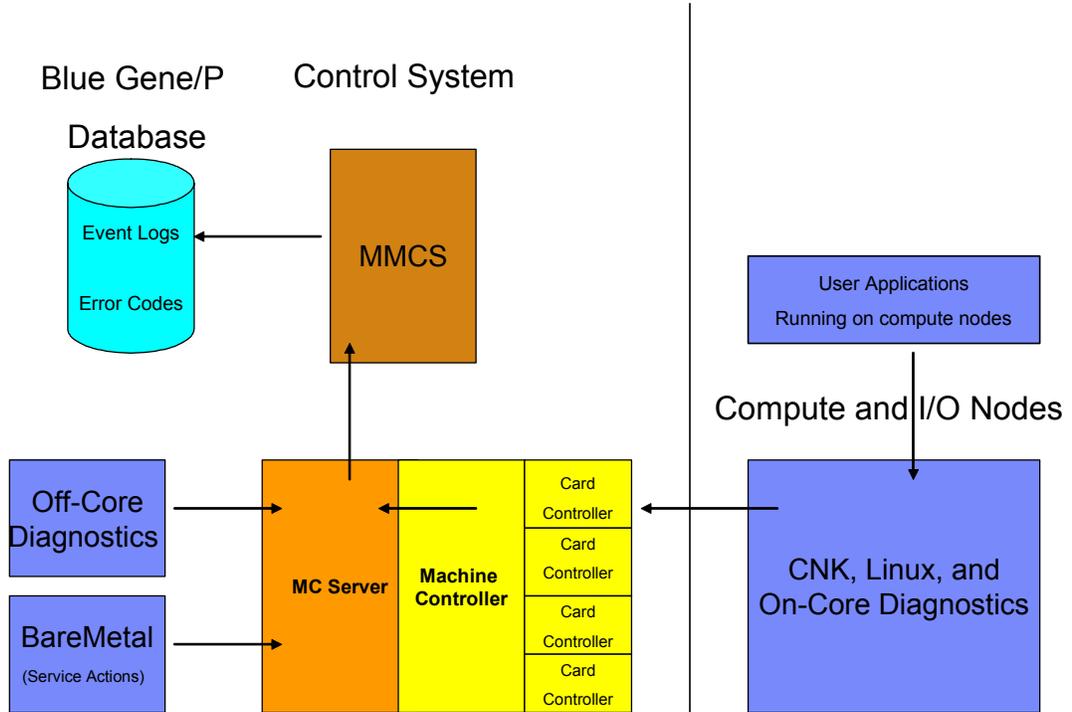


Figure 7-1 RAS Event flow

7.3 Anatomy of a RAS message

Each RAS event message consists of several parts:

- ▶ Message ID
- ▶ Component
- ▶ Subcomponent
- ▶ Error Code
- ▶ Severity
- ▶ Location
- ▶ Message
- ▶ Detailed description
- ▶ Recommended Service Action

Depending on the context you view the message in you may see other information such as the date and time of the event, the number of occurrences or a job ID.

In Figure 7-2 you can see that many of the message parts are column headings in Navigator’s RAS event log. In the Navigator environment you can sort the results by each of the column heads by clicking on the heading itself.

Filter Options: Most recent 50 events (link)					
Event Time	Message ID	Severity	Location	Job ID	Message
4/30/08 7:51:30 AM	KERN_180A	Warning	R20-M1- N09-J04		Environment Monitor threshold warning (Power Calculation Error). Status=0x00000800 PTMON Warnings=0x0x00000000, Temperature=28 C, 1.2v Domain power=460 watts, 1.5v Domain power=115 watts, TotalPower=570 watts EnvError=0x00000004

Figure 7-2 Column heads

Notice the *i* icon in the left most column of the page (Figure 7-2). Clicking on that link will open the details for that specific error. Figure 7-3 shows the message details page after clicking the information icon in Figure 7-2.

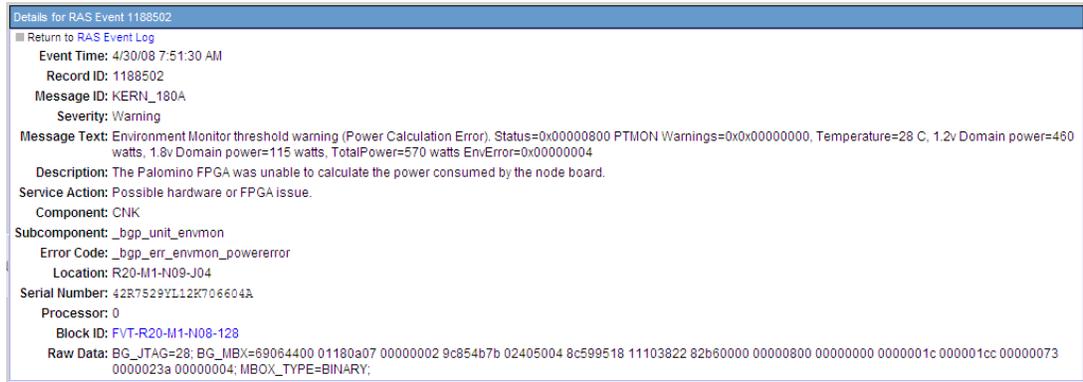


Figure 7-3 Message details

The following sections we will describe the various message part in more detail.

Message ID and components

Each message includes a Message ID, made up of four letters, then an underscore which is followed by four digits. The letters indicate the source of the message, for example a KERN_xxxx originated from the kernel. The following is a list of components that can generate a RAS message and the associated ID:

- ▶ Compute Node Kernel (KERN_xxxx)
- ▶ mcServer (MCTL_xxxx)
- ▶ Diagnostics (DIAG_xxxx)
- ▶ Machine Controller (MCZTL_xxxx)
- ▶ BareMetal (BRMT_xxxx)
- ▶ MMCS (MMCS_xxxx)
- ▶ Card Controller (CARD_xxxx)
- ▶ Application (APPL_xxxx)
- ▶ User-defined (USER_xxxx)

Subcomponent and Error Code

For each of the components above, there are many subcomponents that indicate the functional area that originated the message. Each subcomponent also has error codes that indicate the specific condition of the RAS event. These subcomponents and error codes are stored separately in the RAS database for each RAS message. They are also combined with the component to make up the full message ID described above. The four digits that come after the component in the message ID are determined by the subcomponent (2 digits) and the error code (2 digits).

For instance, the RAS event for Component MMCS, Subcomponent HARDWARE_MONITOR, and Error Code BULK_POWER_WARNING, has a Message ID of MMCS_0206.

Severity

The severity of the event is also included in the message. The possible values are:

- ▶ Info
- ▶ Unknown

- ▶ Error
- ▶ Warning
- ▶ Trace
- ▶ Fatal
- ▶ Debug

Message

The message contains a brief overview of the condition. Depending on the message, the component, location or type of error may be provided.

Detailed description

The event description gives a more general description of the event. The type of information you would typically see in this area would be events that led up to a message. Examples of this would be; environmentals were being read or a specific diagnostic test was running. The *Description* field may also suggest the cause(s) of the problem. For example, the following came from the description area of the message MMCS_0207:

While reading environmental data from the hardware, MMCS detected an error in a card's power module. The error can be related to temperature, current, voltage, or a general fault with the module.

Recommended Service Action

Based on the event, the message may also include a Service Action recommendation. Some errors may always result in the replacement of a compute node, if that is the case the message will indicate that will resolve the problem. In other cases you may be instructed to run diagnostics, or verify a reading on the Environmental Queries page, to determine whether the event is a transient issue.

Other information

Some of the data displayed is dependent on the link you are viewing. For example if you are looking at events in the *RAS Event Log* link you will see the Event Time and possibly a Job ID that is associated with a specific error. If you are viewing the events from the *RAS Message Types* link you will see how many occurrences of each type of event have been logged on your system during the time period selected in the Filter Options.

7.4 Navigator interface to the RAS database

Clicking on the RAS Event Log link in Blue Gene Navigator will open the page shown in Figure 7-4. This interface provides many features that simplify locating, quantifying and analyzing RAS messages.

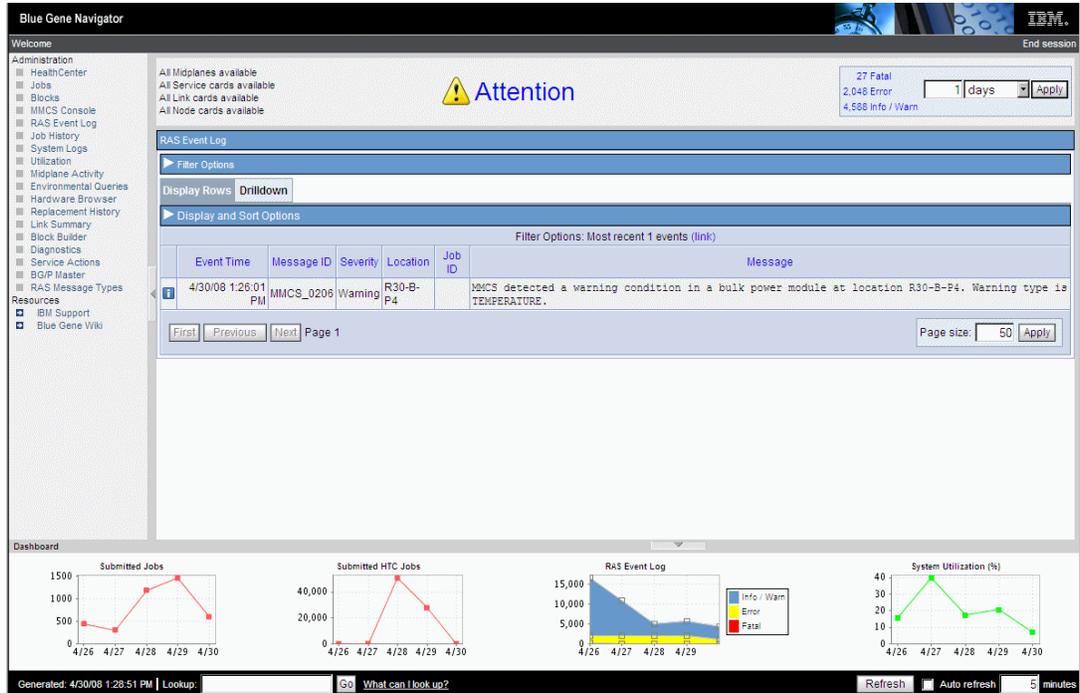


Figure 7-4 RAS Event Log

Filter Options

Figure 7-5 shows the available options when you expand the *Filter Options* link on the RAS Event Log page. The tabs directly under the Filter Options heading allow you to limit the data returned by *Count*, *Relative Time* or *Interval*. The input required on each of the tabs is as follows:

- Count** You supply the maximum number of entries you would like returned. Default is 50
- Relative Time** Provide the number of seconds, minutes, hours or days that you would like to go back in time to search for messages. Default is 30 minutes.
- Interval** Allows you to enter a start and stop time as limits to your search. The default is the preceding 24 hour period.

The common area for each of the tabs allows you to select the various Components and Severities that you can use to filter the data returned. You can further narrow the results by specifying search criteria from the extended filter list such as Message ID, Subcomponent, Error Code and so on. Add the criteria to your search by selecting the box next to the filter and then supplying a value. The *i* icon next to the input area provides you with information such as string limits, wild cards and search options when you hover over it.

Figure 7-5 Filter options

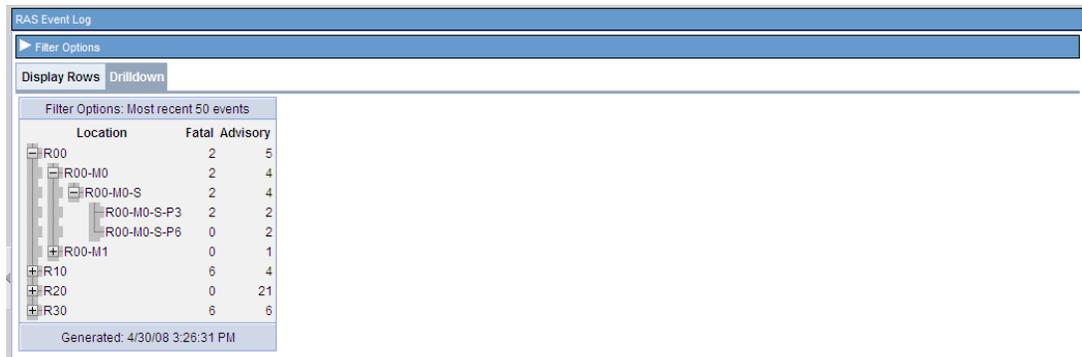
Display Rows versus Drilldown

The default view of RAS Event Log is *Display Rows* which is shown in Figure 7-6. In this view each of the columns can be sorted in ascending or descending order by clicking on the column heading. The information icon to the left will bring up more specific information about the

Event Time	Message ID	Severity	Location	Job ID	Message
4/30/08 3:26:01 PM	KERN_180A	Warning	R20-M1-N09-J04		Environment Monitor threshold warning (Power Calculation Error). Status=0x00000800 PTMON Warnings=0x0x00000000, Temperature=20 C, 1.2v Domain power=460 watts, 1.8v Domain power=115 watts, TotalPower=570 watts EnvError=0x00000004
4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J26		Correctable receiver errors: Node (4,1,1) had 1015306 correctable errors in the X- direction
4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J22		Correctable receiver errors: Node (4,1,0) had 1016868 correctable errors in the X- direction
4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J27		Correctable receiver errors: Node (4,0,1) had 1016108 correctable errors in the X- direction
4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J23		Correctable receiver errors: Node (4,0,0) had 1017621 correctable errors in the X- direction
4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J32		Correctable receiver errors: Node (7,3,1) had 1008274 correctable errors in the X+ direction
4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J28		Correctable receiver errors: Node (7,3,0) had 1008023 correctable errors in the X+ direction
4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J33		Correctable receiver errors: Node (7,2,1) had 1009216 correctable errors in the X+ direction
4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J29		Correctable receiver errors: Node (7,2,0) had 1006701 correctable errors in the X+ direction
4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J24		Correctable receiver errors: Node (4,3,1) had 1002667 correctable errors in the X- direction
4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J20		Correctable receiver errors: Node (4,3,0) had 1003450 correctable errors in the X- direction
4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J25		Correctable receiver errors: Node (4,2,1) had 1001606 correctable errors in the X- direction

Figure 7-6 Display Rows screen

The *Drilldown* tab provides a view of RAS events based on location. Clicking on the + link opens (drills down to) the next level of hardware. This view, for example, may be helpful when tracking down errors related to environmental problems.



The screenshot displays the RAS Event Log interface. At the top, there is a 'Filter Options' section. Below it, two tabs are visible: 'Display Rows' and 'Drilldown', with 'Drilldown' being the active tab. The main content area shows a table with the following data:

Location	Fatal	Advisory
R00	2	5
R00-M0	2	4
R00-M0-S	2	4
R00-M0-S-P3	2	2
R00-M0-S-P6	0	2
R00-M1	0	1
R10	6	4
R20	0	21
R30	6	6

At the bottom of the table, it says 'Generated: 4/30/08 3:26:31 PM'.

Figure 7-7 Drilldown view



8

Diagnostics

This chapter describes the Diagnostics suite that is provided with Blue Gene/P. We demonstrate how to run Diagnostics from the command line and Blue Gene Navigator.

8.1 System Diagnostics

The Blue Gene/P Diagnostic suite interacts with the MMCS server to run tests on the Blue Gene/P hardware. The Diagnostic test suite provides a method of locating hardware failures. These tests aid in categorizing and quantifying hardware failures and determining whether the failures are either an anomaly or systematic.

Diagnostics must be executed from `/bgsys/driver/ppcfloor/bin` on the Service Node or from Blue Gene Navigator. The Navigator is an administrative Web application that you can use to configure and execute Diagnostic runs, view run status, and browse Diagnostic run history. For more information, see Chapter 2, “Blue Gene Navigator” on page 17.

Diagnostics include tests for:

- ▶ Memory subsystem
- ▶ Compute logic
- ▶ Instruction units
- ▶ Floating point units
- ▶ Torus and collective network
- ▶ Internal and external chip communication
- ▶ Global interrupts

Requirements

The Service Node must be running SUSE Linux Enterprise Server 10. The MMCS server and mcServer must be running to control the rack. Your TCP/IP network must be functional because Diagnostics use TCP/IP sockets to communicate between the Service Node and the system.

The Diagnostics suite is installed automatically as a part of the RPM installation of the system. The default directory for the Diagnostics code is `/bgsys/drivers/ppcfloor/diags`. Diagnostic's code is updated with fix packs and Blue Gene/P releases and modifications.

Users who execute diagnostics must have read and execute authority to the `/bgsys/drivers/ppcfloor/diags` directory and the subdirectories that contain the test cases. The output that Diagnostics generates is written to the `/bgsys/logs/BGP/diags` directory. Each time you run Diagnostics, it creates a new directory in the `/bgsys/logs/BGP/diags` directory. These new directories are created using the naming convention `yymmdd_hhmmss_identifier`. Inside the new directories, there is a file named `diags.log` that contains output from the entire run. In addition there are more subdirectories, one for each test case that was included in the Diagnostic run. The subdirectories use the naming convention `TestcaseName_BlockName_timestamp`. Any time a test case runs there is an `output.log` and `rasFile.log` file written to the test case's subdirectory. If the test case completes successfully these logs are deleted in order to save disk space. The user that initiates the Diagnostic run must have all authority to each of these directories.

8.2 Diagnostic test cases

Diagnostics test cases are designed to test all aspects of the hardware. You might have occasions when there is not enough time to run all of the tests. You might also run in to an issue that requires a specific type of test be run, only on certain hardware. To accommodate these instances there are *test buckets*. These test buckets are groups of test cases based on time or a specific type of hardware. When running Diagnostics from the Navigator, you can base the run on the amount of time that you have allowed. The available Navigator test

buckets are small, medium, large, complete and checkup. Running diagnostics from the command line allows you more options. Below is a list of test buckets available on the command line:

checkup	Performs a diagnostics run tailored to be a periodic health check. This bucket makes the assumption that the hardware is for the most part clean and is made to efficiently find and call out degraded hardware. More severe hardware failures will still be detected but the call outs may not be as clear as with some of the buckets listed below
small	A small number of tests to provide the maximum coverage for the minimum runtime.
medium	A medium number of tests to provide the maximum coverage for the minimum runtime.
large	A large number of tests to provide the maximum coverage for the minimum runtime.
complete	All diagnostics are included in this bucket.
servicecard	Runs all service card diagnostics.
nodecard	Runs all diagnostics available for node cards and compute nodes.
linkcard	Runs tests specifically designed for link cards only.
memory	Tests the DDR controllers and DRAM.
ionode	Runs all I/O node specific tests.
multinode	Performs tests on the torus, tree, barrier and DMA functions.
power	Tests the embedded DC-DC power regulators.
global interrupts (gi)	Runs all Global Interrupt diagnostics

Table 8-1 provides a list of the current test cases available, the average amount of run time, a brief description, and the buckets they will run in.

Table 8-1 Test cases

Name	Run time	Description	Bucket
svccardenv	5	This test checks voltages, currents, temperatures, clocks, and other environmental parameters on the service card.	small, medium, large, complete, servicecard, power
readncpower	1	Performs a quick check of the node card power modules.	small, medium, large, complete, nodecard, power
nodecardenv	5	This test checks voltages, currents, temperatures, clocks, and other environmental parameters on the node card.	small, medium, large, complete, nodecard, power
readlcpower	1	Performs a quick check of the link card power modules.	small, medium, large, complete, linkcard, power
linkcardenv	5	This test checks voltages, currents, temperatures, clocks, and other environmental parameters on the link card.	small, medium, large, complete, linkcard, power

Name	Run time	Description	Bucket
bp1lbist	2	This test runs LBIST (Logical Built-In Self-Test) on the BPL ASICs	small, medium, large, complete, linkcard
bpclbist	6	This test runs LBIST (Logical Built-In Self-Test) on the BPC ASICs.	small, medium, large, complete, nodecard
ddr_bitfail	4	Tests the PPC450 by writing to and reading from all DDR memory locations, flagging all failures (data, ECC, chip select, address). The ddr_bitfail test does a simple memory test and prints a log description that attempts to identify the failing component. It identifies specific failing bits by ASIC and DRAM pin when possible.	checkup, small, medium, large, complete, nodecard, memory
ms_gen_short	6	Tests the cache hierarchy within the nodes. This is a more complete memory test that checks both BPC ASIC function and the external DRAM. Error messages in the log attempt to report what failed, but are not always successful.	checkup, small, medium, large, complete, nodecard, memory
conch	10	A memory consistency check.	small, medium, large, complete, nodecard, memory
dgemm_l3	10	This tests the on-chip functionality of the floating point unit on the BPC ASIC.	medium, large, complete, nodecard, memory
dgemm_dram	12	This tests the BPC ASIC, FPU, and memory subsystem on the compute nodes.	large, complete, nodecard, memory, power
dgemm_pulse	14	This tests the BPC ASIC, FPU, and memory subsystem on the compute nodes but it does so through bursts of activity across the nodes, exercising the power modules	checkup, medium, large, complete, nodecard, memory, power
gi_basic	1	Tests the global interrupt network	large, complete, nodecard, gi
gi_comp	1	Tests the global interrupt network	medium, large, complete, nodecard, gi
gi_node	1	Tests the global interrupt network	medium, large, complete, nodecard, gi
gi_link	1	Tests the global interrupt network	large, complete, nodecard, gi
trash	6	Trash runs random instruction streams to exercise the PPC cores.	large, complete, nodecard

Name	Run time	Description	Bucket
tnk	16	TnK is a collection of several tests that exercise both logic and memory paths on the compute nodes.	medium, large, complete, nodecard
tr_diag_loopback	5	This is a single node test of the tree unit involving three major parts: SRAM and its ECC correction, ALU functionality within the tree arbiter, and general packet routing. Any errors found are local to the ASIC.	medium, large, complete, nodecard, multinode
ts_dma_loopback	5	This tests the DMA loopback.	checkup, medium, large, complete, nodecard, multinode
tr_and_dma_connectivity	8	This tests the tree and DMA connectivity.	checkup, medium, large, complete, nodecard, multinode
XEMAC_loopback_polling	5	The XEMAC is configured to be in internal loopback mode. This test transmits a single TCP packet, polls the Received Frames register for successful reception of the packet and verifies the content of the received packet. Both the transmission and reception are done by a single core.	medium, large, complete, nodecard, ionode
XEMAC_loopback_interrupt	5	The XEMAC is configured to be in internal loopback mode. This test transmits a single TCP packet, receives an interrupt that a packet has been received and verifies the content of the received packet. The transmission is done by core 0 and the reception is done by core 1.	medium, large, complete, nodecard, ionode
EMAC_loopback_multi_core	5	The XEMAC is configured to be in internal loopback mode. A stream of packets are transmitted by one core and received by another core. The Received Frames register is polled for successful reception of the received packets.	medium, large, complete, nodecard, ionode
PHY_loopback	5	In this, the external PHY chip (from Broadcom) is configured to be in loop back mode. A stream of packets are transmitted by one core and received by another core. The Received Frames register is polled for successful reception of the received packets. This tests the entire path: TOMAL0 → XEMAC0 → XGXSPCS0 → PHY0 → XGXSPCS0 → XEMAC0 → TOMAL0.	medium, large, complete, nodecard, ionode

Name	Run time	Description	Bucket
node2node_multi_packet	5	In this test, a stream of packets are transmitted by one node and received by another node. This tests the node to node path: TOMAL0 → XEMAC0 → XGXSPCS0 → PHY0 → optical link → PHY1 → XGXSPCS1 → XEMAC1 → TOMAL1. This test requires the node card XFPs and optical cables be plugged in.	large, complete, nodecard, ionode
neighbor	4	A basic torus test. Sequentially, each node sends a message to its six neighbors. Hangs in the test indicate torus communication problems, which is how the test determines bad torus links.	checkup, complete, nodecard, multinode

8.3 Running Diagnostics

As we mentioned earlier in this chapter, there are two methods to initiate Diagnostics. The first approach is to use Blue Gene Navigator. The second method is to start them from a command line on the Service Node. To run Diagnostics, you must have the authorities listed in Chapter 8.1, “System Diagnostics” on page 126.

8.3.1 Navigator’s Diagnostics interface

Clicking the Diagnostics link in the navigation pane of the Navigator opens the page shown in Figure 8-1. The most recent runs for midplane, link, and user-defined block Diagnostics are listed on this page. Each Location or Block ID is a link that takes you to page that contains the history of all Diagnostic runs for that specific location or Block ID.

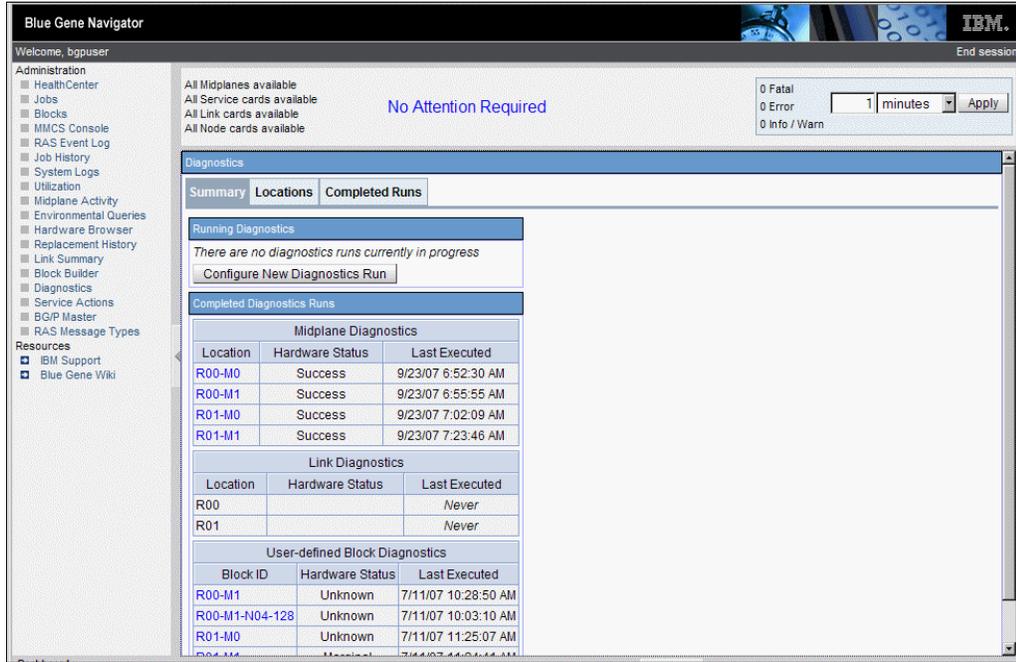


Figure 8-1 Diagnostics home page

Each Location or Block ID is a link that takes you to page that contains the history of all Diagnostic runs for that specific location or Block ID. We selected the R00-M1-N04-128 user-defined block link. That opens the page shown in Figure 8-2. By clicking the date and time stamp link on the left, you can view the result for each test case that was included in the run. In our example there was only one test case in the run, ts_dma_loopback.

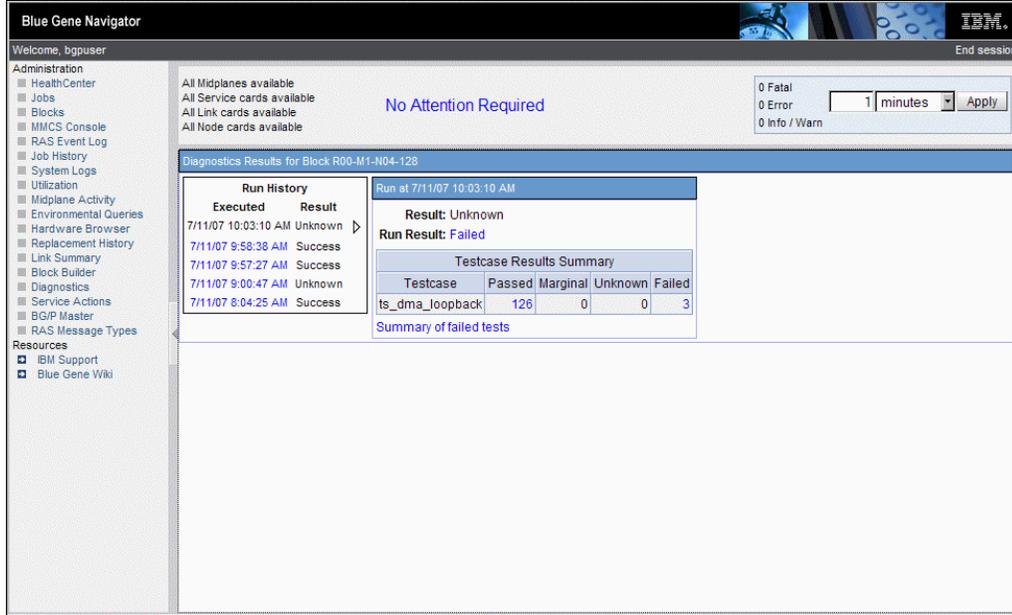


Figure 8-2 User-defined block page

Back on the Diagnostics main page there are three tabs across the top of the main content area. The Summary tab is the page you see when first clicking the Diagnostics link. The Locations tab, shown in Figure 8-3, provides you with a view of all the hardware that has had Diagnostics run on it. Each location string on the left is a link to the summary of diagnostics that have been run on that location. You can filter the results on, and sort by, each of the column heads.

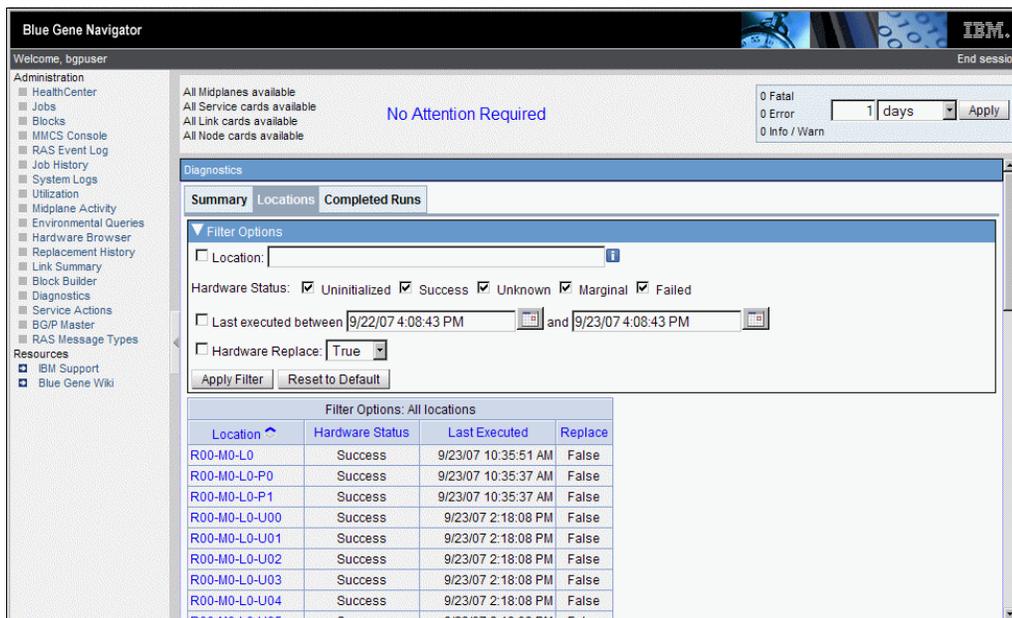


Figure 8-3 Diagnostics Location tab

The Completed Runs tab, Figure 8-4, shows a history of all the runs of Diagnostics that have run to completion. Included on the page are Filter Options that allow you to trim the results shown on the page according to the status, time of run, and the target of the run (midplane, rack, or block).

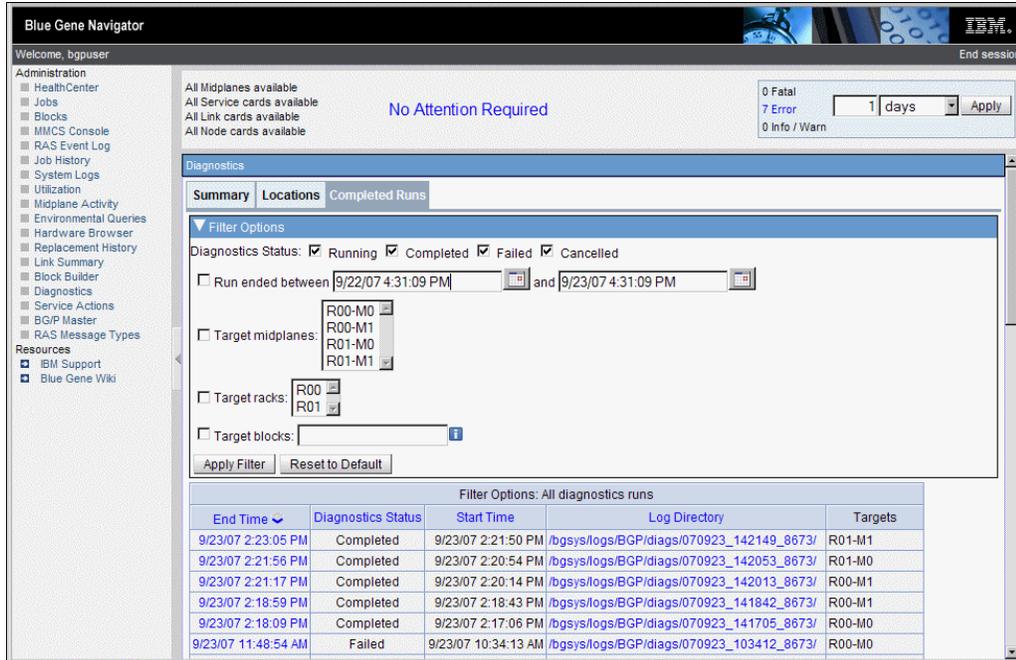


Figure 8-4 Completed runs tab

To get more information about a specific run, you can click the link in the Log Directory column, which opens the log files for that run. To see the details for a specific run, click the link in the End Time column. See Figure 8-5. Click **Collect Logs for Run** at the bottom of the page to generate a tar file with the logs from the run. The tar file is generated in the /bgsys/logs/BGP/diags directory.

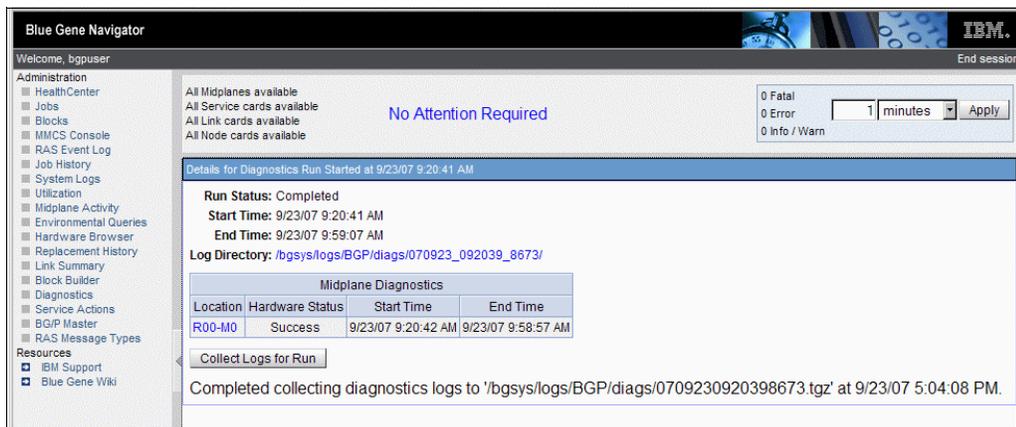


Figure 8-5 Diagnostics run details

8.3.2 Starting a Diagnostic run

To start a Diagnostics run from the Blue Gene Navigator, follow these steps:

1. Go to the Summary tab. Under the Running Diagnostics heading, click **Configure New Diagnostics Run**. This opens the page shown in Figure 8-6.

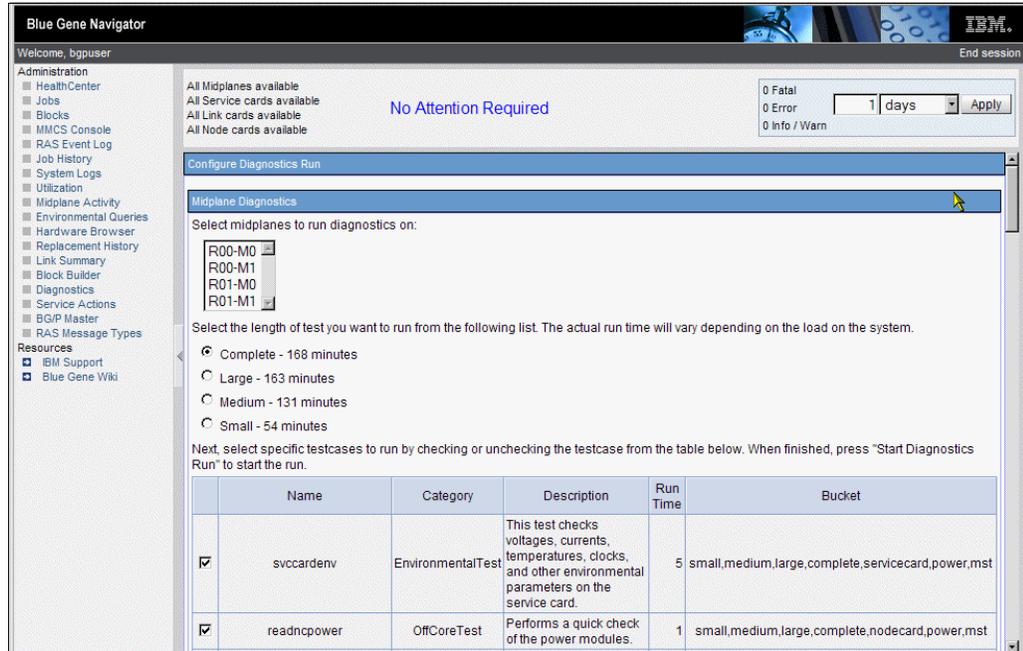


Figure 8-6 Configure Diagnostic run

2. Select the midplanes that you want to include in the run. You can select multiple midplanes by holding the Ctrl key and left clicking each of the midplanes on which you want to run diagnostics.
3. Next select the length of test that you want to run: Checkup, Complete, Large, Medium, or Small.
4. You can add or remove test cases individually by toggling the check mark on or off in the box on the left-hand side of the page.
5. At the bottom of the page, you can select the Run Options. See Figure 8-7.
 - Pset ratio override: By default the system configures the run to use all of the I/O nodes on the system. If you have a system that does not have all of the I/O nodes plugged in to the ethernet, you want to change this to reflect the I/O node to compute node ratio of your system. To override the system, click the box to insert a check mark. Then from the drop-down menu, select the ratio that you want to use.
 - If you want the Diagnostics run to stop on the first error it encounters, toggle the check mark on in the second box.
 - Placing a check mark in the third box forces Diagnostics to save all the output from the run.
 - Midplanes per row: by default diagnostics will only allow two midplanes to be run in parallel within a row of the machine. If more are desired you can place a check mark in the box and enter the desired number of concurrent midplane runs in the text box.
 - Midplanes per service node processor: by default diagnostics will only allow two midplanes to be run in parallel per processor in the service node. If more are desired

you can place a check mark in the box and enter the desired number of concurrent midplane runs in the text box.

6. To start the run, click **Start Diagnostics Run** or click **Cancel** to abort the run. A message prompts you to verify that you want to start the Diagnostic Run. Click **OK** to continue.

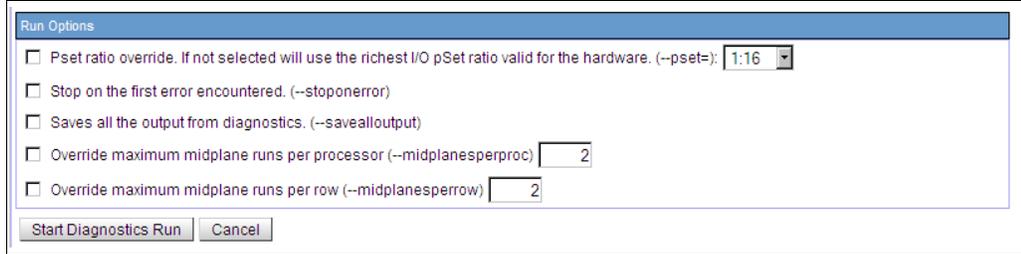


Figure 8-7 Run Options

After the run has started, you are sent back to the Summary tab. There, you see a list of all the midplanes configured to run in the Running Diagnostics box. See Figure 8-8. You can cancel the run at any time by clicking **Cancel** next to the start time of the run.

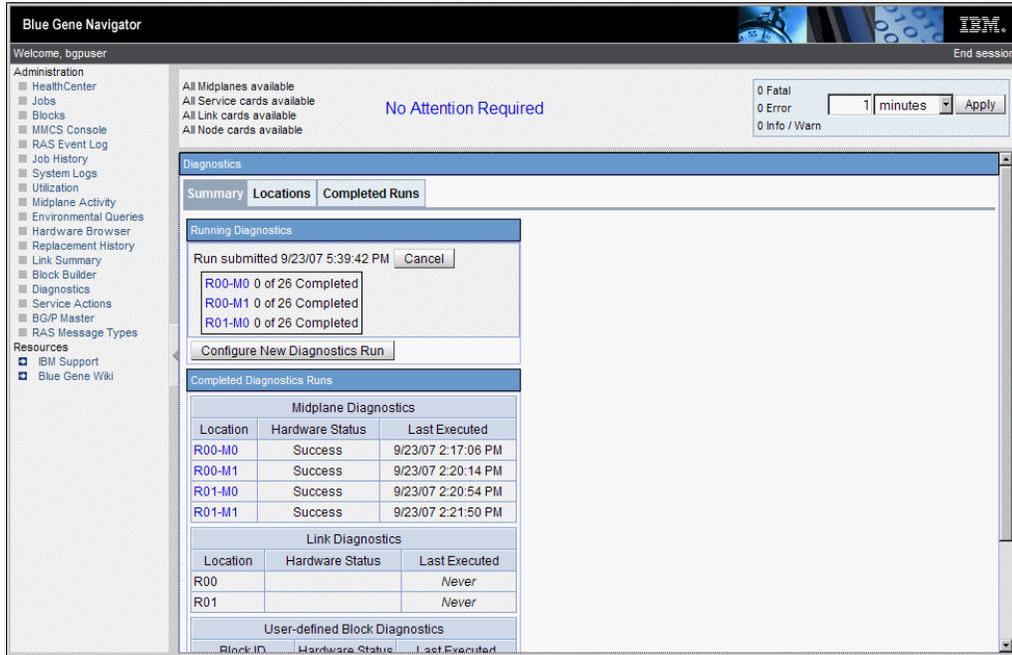


Figure 8-8 Diagnostic run started

Click the midplane link to view each of the test cases, and the results, that are configured to run. See Figure 8-9. The number of test cases run and the status of the run are updated continually as the Diagnostic run progresses. Use the Refresh button at the bottom of the page to update the status of the run. You can set the Auto refresh feature by entering the number of minutes that you want between refreshes and then selecting the box to toggle on the check mark.

The screenshot shows the Blue Gene Navigator interface. The main window displays a 'Testcase Results Summary' table. The table has columns for Testcase, Started, Finished, Passed, Marginal, Unknown, and Failed. The test cases listed include PHY_loopback, XEMAC_loopback_interrupt, XEMAC_loopback_multi_core, XEMAC_loopback_polling, bpcibist, bpllbist, conch, ddr_bifail, dgemm_dram, dgemm_l3, dgemm_pulse, gi_basic, gi_comp, gi_link, gi_node, linkcardenv, ms_gen_short, neighbor, nodecardenv, readlcpower, readncpower, and svccardenv. The status of each test case is indicated in the 'Finished' column, with some showing 'Running' or 'Failed'.

Testcase	Started	Finished	Passed	Marginal	Unknown	Failed
PHY_loopback						
XEMAC_loopback_interrupt						
XEMAC_loopback_multi_core						
XEMAC_loopback_polling						
bpcibist	9/23/07 6:04:30 PM	Running				
bpllbist	9/23/07 6:03:29 PM	9/23/07 6:04:30 PM	24	0	0	0
conch						
ddr_bifail						
dgemm_dram						
dgemm_l3						
dgemm_pulse						
gi_basic						
gi_comp						
gi_link						
gi_node						
linkcardenv	9/23/07 6:03:15 PM	9/23/07 6:03:29 PM	4	0	0	0
ms_gen_short						
neighbor						
nodecardenv	9/23/07 6:02:50 PM	9/23/07 6:03:06 PM	16	0	0	0
readlcpower	9/23/07 6:03:07 PM	9/23/07 6:03:15 PM	8	0	0	0
readncpower	9/23/07 6:02:14 PM	9/23/07 6:02:50 PM	128	0	0	0
svccardenv	9/23/07 6:01:54 PM	9/23/07 6:02:14 PM	1	0	0	0

Figure 8-9 Test case details

8.3.3 Running Diagnostics from the command line

The second method of initiating a Diagnostics run is from the command line. The `rundiags.py` command is located in the `/bgsys/drivers/ppcfloor/bin` directory. An example of the usage is:

```
$ /bgsys/drivers/ppcfloor/bin/rundiags.py --midplanes R00-M0,R01-M1
```

This example runs the Complete Diagnostic bucket on midplanes R00-M0 and R01-M1.

The command requires that you specify either midplane, racks, or blocks to run the diagnostic test cases on. The syntax for those parameters are:

--midplanes x,y,z... List of midplanes on which to run diagnostics (for example, R00-M0,R00-M1,R01-M0)

--racks x,y,z... List of racks on which to run diagnostics. (for example, R00, R01, and R10)

--blocks x,y,z... List of blocks on which to run diagnostics

The list of midplanes, racks, or blocks is a comma separated list and must not contain any whitespaces. The **--blocks** can only be specified by itself. It is not compatible with the **--rack** or **--midplanes** switches. Midplanes and racks can be used in the same command, for example:

```
$ /bgsys/drivers/ppcfloor/bin/rundiags.py --midplanes midplane_list --racks rack_list [OPTIONS ... ]
```

Table 8-2 shows the options that can be used with the `rundiags.py` command.

Table 8-2 The `rundiags` options

Option	Default	Description
<code>--bootoptions</code>	mergepers	Manually specify boot options
<code>--bringupoptions</code>	fischer_connector_clock	Manually specify bringup options
<code>--buckets</code>	Complete	Runs the specified bucket, (See 8.2, "Diagnostic test cases" on page 126)
<code>--cloc</code>	Not used by default	The location string to use on connect requests
<code>--createxmlresult</code>	false	Indicates if the results.xml file should be created
<code>--csport</code>	32031	Control system port
<code>--csuser</code>	current user	Control system user
<code>--dbname</code>	bgdb0	Name of the database (see the db.properties file)
<code>--dbpassword</code>	xxxxxx	Database password (see db.properties file)
<code>--dbport</code>	50001	Port that DB2 is listening on
<code>--dbproperties</code>	/bgsys/local/etc/db.properties	db.properties file
<code>--dbschema</code>	bgpsysdb	Name of the database schema (see db.properties file)
<code>--dbuser</code>	bgpsysdb	Name of the database user (see db.properties file)
<code>--floor</code>	/bgsys/drivers/ppcfloor	Specify an alternate floor directory
<code>--forceblockfree</code>	false	Forces a free of the block prior to use.
<code>--help</code>		Displays help text and exits
<code>--logdirprefix</code>	None	Add a user-defined prefix to the log directory
<code>--mcserverport</code>	1206	mcServer port
<code>--midplanesperproc</code>	2	The number of concurrent midplane runs allowed per service node processor
<code>--midplanesperrow</code>	2	The number of concurrent midplane runs allowed per row.
<code>--nogenblock</code>	false	Prevents Diagnostics from automatically creating blocks
<code>--outputdir</code>	/bgsys/logs	Overrides the default output directory
<code>--pid</code>	PID	Specifies the process ID to assume when generating the run ID
<code>--pset</code>	Maximum I/O to compute node ration on the system	Overrides the default pset ratio
<code>--savealloutput</code>	false	Saves all the output generated by Diagnostics

Option	Default	Description
--sn	localhost	Name of the Service Node
--stoponerror	false	Will cause the run to abort when the first error is encountered
--tests	Run all tests	This option allows you to provide a list of tests to be included in the run. Can be used in conjunction with --buckets to run additional test cases.
--testxml	test.xml	Diagnostics tests file
--verbosity	0	Verbosity level of output generated by Diagnostics



9

Service Actions

Preparing the Blue Gene/P hardware for service or for a system power cycle is a normal part of the System Administrator's duties. In this chapter, we describe Service Actions and discuss how to prepare the hardware for service and how to cycle power.

9.1 Service Actions overview

There will be occasions when you need to cycle power or replace hardware on your Blue Gene/P system. Any time service on the Blue Gene/P hardware is to be performed, the hardware needs to be prepared by generating a *Service Action*.

Service Actions are implemented in one of the following ways:

- ▶ Through the Blue Gene Navigator
- ▶ Directly on the Service Node's command line

The following list contains the hardware that can be specified in a Service Action:

- ▶ Single Node card
- ▶ Multiple Node cards on a midplane
- ▶ Link card
- ▶ Any or all cards in a midplane (except the Service card)
- ▶ Fan Module
- ▶ Bulk Power Module
- ▶ Rack (allows the Service card to be removed)
- ▶ Clock card

As a general rule, if the hardware can be removed with your fingers, then you can simply generate a Service Action for that location prior to removing the part. If the service to be performed requires the use of tools, then you need to turn off the power to the rack before removing it.

Attention: At least one functioning Bulk Power Module (BPM) must be plugged in at all times during a Service Action so five volt (5V) persistent power is maintained to the clock card. If the power to the rack is switched off, the clock cards will lose persistent power.

When a Service Action is initiated, there are a few internal procedures and checks that occur that are common to most hardware prior to preparing it for service:

- ▶ A check is done to ensure that there are no conflicting Service Actions in progress.
- ▶ A Reliability, Availability, and Serviceability (RAS) event is written to the event logs stating that a Service Action has been started on a specific piece of hardware along with the name of the person that initiated the action.
- ▶ Jobs that are using affected hardware are ended and blocks are freed.
- ▶ Job schedulers are prevented from allocating resources that are marked for Service Actions.
- ▶ Vital Product Data (VPD) is written to the database.
- ▶ Some hardware communicates to the Service card through an Ethernet port on the midplane. When a Service Action is initiated on that type of hardware, its Ethernet port on the Service card's switch is disabled.
- ▶ The hardware's status is updated in the database.
- ▶ A RAS event is posted indicating that the hardware is ready for service.

9.1.1 Service Action logs

The Service Action logs are stored in the `/bgsys/logs/BGP` directory. You can access the logs from the command line or from the System Logs link in Blue Gene Navigator. The naming format used follows the syntax `<service action>-location-timestamp.log`, where `<service`

action> correlates to the Servicexxx command that was used to initiate the action. For example, a Service Action started for node card R01-M0-N04 on 07/06/2007 at 13:32:25 is named *ServiceNodeCard-R01-M0-N04-2007-07-06-13:32:25.log*. When that Service Action is ended, you see another log with a similar name, but the time stamp will be different. In the previous example, if the Service Action after 11 minutes, the log file name is *ServiceNodeCard-R01-M0-N04-2007-07-06-13:43:25.log*.

Important: You can run only one Service Action on any specific component at a time. For example, if there is an open Service Action on R00-M1-N05, you are not allowed to run a PrepareForService on R00-M1.

Evolution Note: Replacing a Service card in Blue Gene/L requires that you turn off the power to that rack. With Blue Gene/P, you can replace a Service card without turning off the power to the rack using the ServiceMidplane, ServiceRack, or ServiceClockCard command.

9.2 Preparing the hardware for service

As mentioned previously, there are two ways to initiate a Service Action: from the command line or from the Navigator. The only difference between the two is that you cannot supply the optional parameters when submitting the Service command from the Navigator. In this section, we detail the steps for both methods.

9.2.1 Command line Service Actions

The commands used to start, end, and close Service Actions are located in the `/bgsys/drivers/ppcfloor/bareMetal` directory. The commands are:

- ▶ ServiceBulkPowerModule
- ▶ ServiceFanModule
- ▶ ServiceLinkCard
- ▶ ServiceMidplane
- ▶ ServiceNodeCard
- ▶ ServiceRack
- ▶ ServiceClockCard

Important: To end the ServiceRack Service Action you must remove and reseal one of the BulkPowerModules. You will need physical access to the rack to complete a ServiceRack.

You need to qualify each of the commands with a location. For example, the ServiceNodeCard command requires that you give the location of a specific Node card, such as R00-M0-N04, or all the Node cards in a midplane, R00-M0-N. In addition to the location, you also need to specify the action.

The list of possible actions are:

Prepare	Prepare the card for service
End	End the Service Action and make the card functional
Close	Force an existing open or prepared Service Action to the closed state

There are several optional parameters that can be supplied:

- user <userID>** The user name to be specified when connecting to mcServer. If a user is not specified the default is the user who issued the command.
- msglevel** Controls the output message level.
 Verbose: Detailed output messages
 Debug: Includes all of the output from Verbose
- dbproperties <path>** The fully qualified file name for the db.properties file. The default is /bgsys/local/etc/db.properties.
- base <path>** The base (install) path to the files that are used by this command. The default is /bgsys/drivers/ppcfloor.
- log <path>** The path to the directory that contains the log file from this command. The default is /bgsys/logs/BGP.
- help** Displays the help text for the command.

An example of the syntax to prepare R00-M0-N4 for service from the command line is:

```
./ServiceNodeCard R00-M0-N04 Prepare
```

9.2.2 Starting Service Actions from the Navigator

In Blue Gene Navigator clicking the Service Action link in the navigation pane takes you to the page shown in Figure 9-1. In addition to initiating Service Actions from this page, you can also filter your view to show Service Actions, historical, and current. You can also query by time, location, or user.

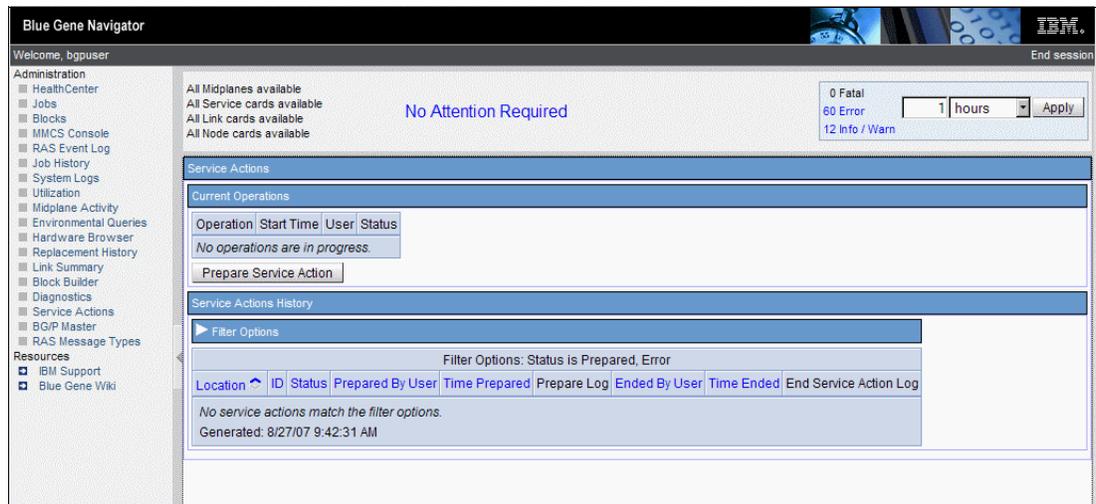


Figure 9-1 Service Actions link in Navigator

Figure 9-2 shows the possible filters that can be applied to query the *bgpserviceaction* table.

Filter Options

Status: Actively Processing Ended Error Forced Closed Open Prepared

Prepared between 8/27/07 8:45:11 AM and 8/28/07 8:45:11 AM

For location

Prepared by user

Ended by user

Apply Filter Reset to Default

Figure 9-2 Service Action filters

Starting a Service Action

To start a Service Action, follow these steps:

1. Click **Prepare Service Action** in the content area of the page to open the page shown in Figure 9-3. From this window, you can choose the hardware that you want to service. Click **Next** to proceed.

Blue Gene Navigator

Welcome, bgpuser

Administration

- HealthCenter
- Jobs
- Blocks
- MMCS Console
- RAS Event Log
- Job History
- System Logs
- Utilization
- Midplane Activity
- Environmental Queries
- Hardware Browser
- Replacement History
- Link Summary
- Block Builder
- Diagnostics
- Service Actions
- BGP Master
- RAS Message Types

Resources

- IBM Support
- Blue Gene Wiki

All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Fatal
60 Error
12 Info / Warn

1 hours Apply

Prepare Service Action

Pick the type of service action:

- Service a node card
- Service the node cards on a midplane
- Service a link card
- Service the cards on a midplane
- Service a fan module
- Service a bulk power module
- Service a rack
- Service a clock card

Click next to select the target of the service action.

Back Next Finish Cancel

Figure 9-3 Start a Service Action

- The next page that opens allows you to select a target location for the Service Action (Figure 9-4). Recall that starting a Service Action on hardware that has a block booted or job running causes that block to be freed or job to be terminated. Click **Next** to verify that the hardware that you have selected is not in conflict with any running jobs.

Note: If you are performing a Service Action on a fan module or a bulk power module, the Next button is disabled. These types of Service Actions do not affect jobs.

If you are not concerned about potential conflicts, you can click **Finish** from this page and proceed with the Service Action.

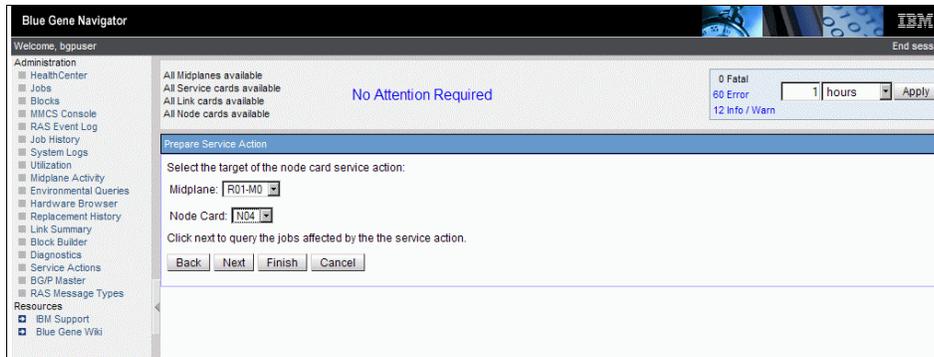


Figure 9-4 Select the target

Another issue that could potentially arise is if you attempt to start a Service Action for a specific component that a pending Service Action already exists. See Figure 9-5.

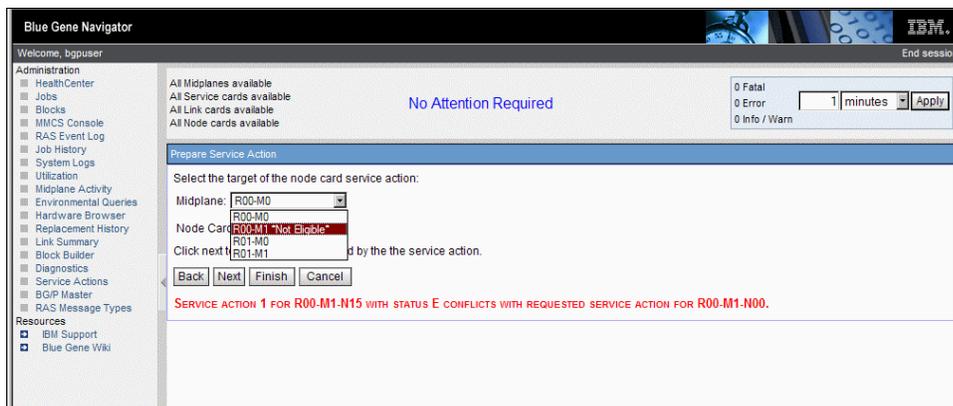


Figure 9-5 Conflicting Service Actions

Messages in the drop-down menu and within the window also alert you to possible conflicts with existing Service Actions. When you are satisfied with your choices, you can click **Finish** to start the Service Action (Figure 9-6).

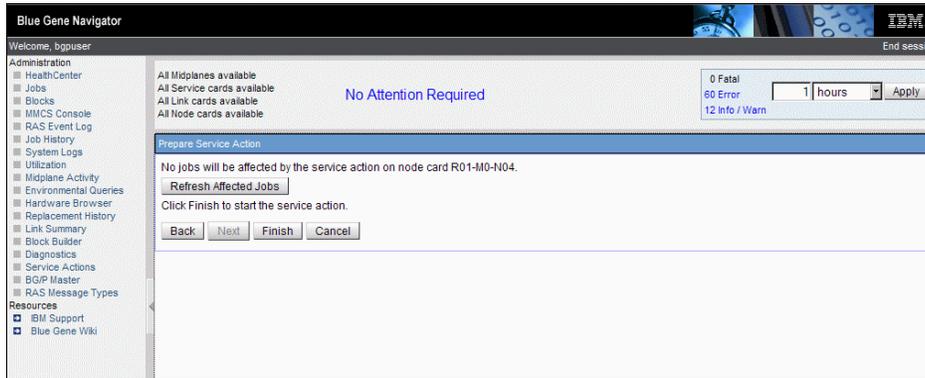


Figure 9-6 Check for conflicts

3. A window prompts you to verify that you want to start the Service Action. Click **OK** to proceed.
4. You see a screen similar to Figure 9-7. If you click **Refresh** at the bottom of the window you see the status change as the Service Action progresses.

Tip: In most cases, you need to use the Refresh button at the bottom of the Navigator window. Because of the type of data that is contained in the pages (POSTDATA), using the browser's refresh feature can cause errors.

When running a Service Action on a node card, you see the following status changes:

- Request node card information
- Starting
- Issuing write VPD (the card becomes unavailable at this point)
- Issuing power off request for node card Rxx-My-Nzz

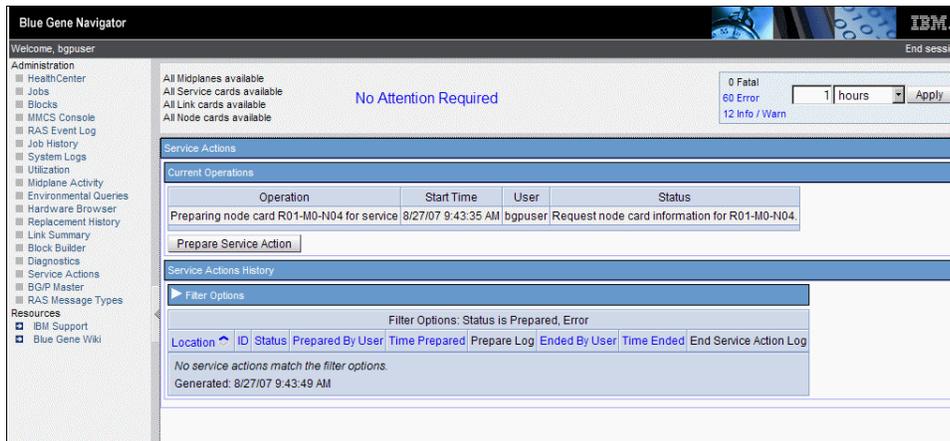


Figure 9-7 Initial Service Action page

5. When the specified hardware is ready for service, you see the Service Action window change to the one shown in Figure 9-8. Note the status, in this case *Prepared*, which indicates that the prepare Service Action was successful. It is important to verify the status prior to performing any service on the system. For example, if the status is *Error*, the power might not have been properly terminated, and removing hardware at this point could be hazardous not only to the hardware itself, but also to the service personnel.

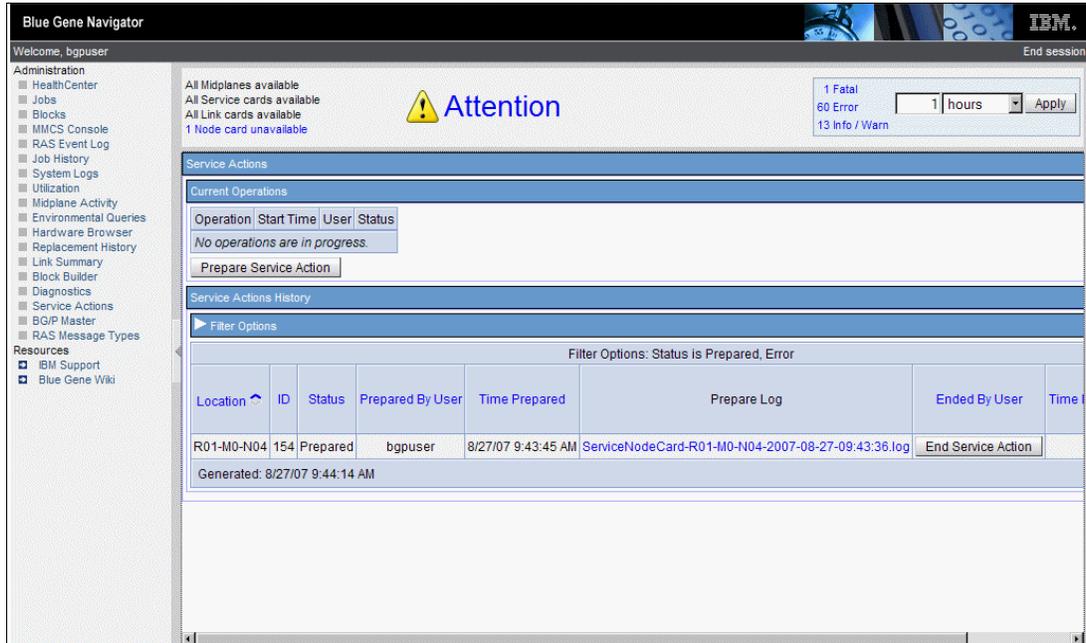


Figure 9-8 Service Action ready

Ending a Service Action

While the Service Action is open, you see the page shown in Figure 9-8. To return the hardware to a usable state, click the corresponding **End Service Action** button for the Service Action that you want to complete. You are prompted to verify the action. Click **OK**.

In the Current Operations section of the main content area, you see the Status column make the following changes as the hardware is re-initialized:

1. Issuing initialize node card request.
2. Request node card information.
3. Initialize node card Rxx-My-Nzz request completed (the hardware will become available at this point).

9.3 Cycling power

The process to cycle power on your system depends on the reason you are shutting down the system and the number of Blue Gene/P racks in your system. Any time you are replacing parts within an individual rack you should run a Service Action to prepare the hardware for service. When the Service Action is ended the database will be updated to reflect the new hardware that was installed. If you are shutting down the system with no intentions of replacing hardware, or to replace hardware in multiple racks, there is an alternative process that we describe in this section.

Important: When ending a Service Action the system is surveyed to find any new hardware. Replacing hardware without a Service Action will cause the database to be out of sync with the hardware that is actually installed in the system.

Single rack system

Prior to turning power off on a rack, you need to prepare the rack properly. Preparing a rack to turn off the power requires that you run the ServiceRack command. The steps to power off a single rack are:

1. First prepare the rack for service from either the Navigator or the command line. The command line syntax to start a service action on a rack is:

```
$ ServiceRack Rxx PREPARE
```

To prepare a rack for service from the Navigator interface follow the steps listed in 9.2.2, “Starting Service Actions from the Navigator” on page 142. In step number one select the *Service a rack* option (see Figure 9-3 on page 143).

2. The ServiceRack process terminates any jobs that are running on that specific rack and powers down all the hardware in the rack with the exception of the clock card. After the ServiceRack command completes, you receive a message that indicates that the rack is ready for service. At this time you can power the rack off.

If you don't power off the rack you must reseal one of the Bulk Power Modules (BPM) to bring the power back on line before ending the Service Action. Reseating a BPM provides the master Service Card with power which will then power up the remaining Bulk Power Modules.

3. Then end the Service Action from the command line using the following command:

```
$ ServiceRack Rxx END
```

To end the Service Action from Blue Gene Navigator, click the *End Service Action* button that corresponds to your current action.

Multiple rack system

There are two possible scenarios that we discuss in this section. The first is a system power down with no hardware to replace and the second is a power down with hardware replacement. In either scenario you could use the *ServiceClockCard* command on the rack that contains the master clock. The advantage of this method is that all of the jobs on the system will be killed and when the Service Action is ended all of the link cables will be verified. Keep in mind however that ending this type of service action only verifies new hardware on the rack that contains the master clock.

Option 1

In this scenario we choose not to use the ServiceClockCard command. The steps to power down a multiple rack system when there is no hardware to be replaced are:

1. Stop the BGP Master process using the following command:

```
$ ./bgpmaster stop
```

2. Power off all of the racks.

To bring the system back on line use the following steps:

1. Power on the racks starting with the rack that contains the master clock followed by the secondaries, tertiaries and then the remaining slave racks.
2. Start the BGP Master processes with the following command:

```
$ ./bgpmaster start
```

Option 2

The second option for powering off the system may involve replacing hardware. The steps for this scenario are similar to “Option 1” but in order to discover new hardware we will run an Install Service Action (ISA) once the system is restarted.

1. Stop the BGP Master process using the following command:

```
$ ./bgpmaster stop
```

2. Power off all of the racks.

After the hardware has been replaced and you are ready to bring the system back on line use the following steps:

1. Power on the racks starting with the rack that contains the master clock followed by the secondaries, tertiaries and then the remaining slave racks.
2. Start the BGP Master processes with the following command:

```
$ ./bgpmaster start
```

The MMCS Server should not be running during the ISA, run the following command to stop it:

```
$ ./bgpmaster stop mmcs_server
```

3. From the /bgsys/drivers/ppcfloor/baremetal directory start the ISA:

```
$ ./InstallServiceAction
```

4. If you replaced a Link card you should also verify that the cabling is correct. From the /bgsys/drivers/ppcfloor/baremetal use the following command:

```
$ ./VerifyCables
```

5. Stop and restart the BGP Master process.

```
$ ./bgpmaster stop
```

```
$ ./bgpmaster start
```



Blue Gene/P tools

This chapter describes tools that are provided with Blue Gene/P and includes sections on the Security Administration tool and the Coreprocessor tool.

10.1 Security Administration tool

The Security Administration tool provides a means of assigning authority consistently to the users who access the Blue Gene system. The tool authorizes users to various predetermined functions on the system by adding their profile to a selected group. The predefined groups are:

- ▶ bgpuser
- ▶ bgpdeveloper
- ▶ bgpservice
- ▶ bgpadmin

The groups are created on the Service Node when the Blue Gene/P system is installed. You can edit the program to define the groups differently.

Existing Linux users are added to groups by running the bguser.pl utility. The syntax for the command is:

```
./bguser.pl [options]
```

Options are:

- ▶ **--user** *userName*
- ▶ **--role** [*user/developer/service/admin*]

Table 10-1 provides a list of the available roles and capabilities that are allowed when a user is added using the tool.

Table 10-1 The bguser roles

Role	Capability
user	<ul style="list-style-type: none"> ▶ submit Jobs through mpirun ▶ read access to small amount of data (job/block status) on Service node, through Navigator ▶ access to the Front End nodes ▶ complete access compilers/tool chain/etc for development on the Front End nodes
developer	<ul style="list-style-type: none"> ▶ submit jobs through mpirun ▶ read access to some data (job/block status) on Service node, through Navigator ▶ Controlled and limited access to Service node - requires user ID on SN ▶ Doesn't have root access but has elevated privileges ▶ complete access compilers/tool chain/etc for development on the Front End nodes. ▶ debugs with coreprocessor
admin	<ul style="list-style-type: none"> ▶ complete access to Blue Gene/P functions on the Service Node and Front End Node(s)
service (IBM)	<ul style="list-style-type: none"> ▶ access to required debug tools, system logs, read access to database

10.2 Coreprocessor tool

The Coreprocessor tool is a basic parallel debugger that enables parallel debug of problems at all levels: hardware, kernel, and application. The Coreprocessor tool uses the low-level hardware JTAG interface to read and organize hardware information, including instruction address registers (IAR), general purpose registers (GPR), special purpose registers (SPR), and device control registers (DCR). It can process compute node core files, and it can also connect to a running job.

The Coreprocessor tool levies no dependencies on the application code that is running on the node (no special calls that need to be made, libraries to link in, and so on). It can sort nodes based on their stack traceback and kernel status, which can help isolate a failing or problem node quickly. It also supports stack dumping on a per processor basis.

The tool would most likely be used:

- ▶ To examine compute node core files. For performance and disk space reasons, these core files are simple text files, and their format is not understood by all debuggers. The Coreprocessor tool points out the node that is acting in a suspicious manner and probably caused the partition to dump.
- ▶ When your usual debugger is not able to connect to the compute nodes. Other debuggers are not useful when a node is non-responsive. However, because the Coreprocessor tool uses the JTAG connection to a node to collect debug information, it does not require a functional connection to a compute node through an I/O Node.

Some of the advantages of using Coreprocessor tool are:

- ▶ OS can be completely dead and can still handle debug
- ▶ Quick isolation of nodes that are abnormal
- ▶ Scales to a full Blue Gene/P system

With the Coreprocessor tool, you cannot set breakpoints, and there is no visibility. However, it does provide the instruction address and function name. It also provides the line number when the code has been compiled with the debug option.

Dependencies

The PERL modules used by the Coreprocessor tool are:

- ▶ `cwd`
- ▶ `Compress::Zlib`
- ▶ `IO::Socket`
- ▶ `OpenGL` (for 3D Map function only)
- ▶ `Tk`

Note: Tk is not usually installed with the operating system. One way to install it is to run the following command:

```
sudo perl -MCPAN -e "install Tk" \;
```

Answer *no* at the prompt to let it autoconfigure.

10.2.1 Starting the Coreprocessor tool

The Coreprocessor tool is initiated from a VNC session. From the command line (VNC), enter the following command:

```
/bgsys/drivers/ppcfloor/tools/coreprocessor/coreprocessor.pl
```

There are no required parameters to start the tool. Table 10-2 provides a list of options that you can use when starting the tool.

Table 10-2 Options to use when starting the tool

Option	Description
-b	Specifies the ELF image to load for compute nodes. Multiple ELF files can be specified by concatenating them with a colon ':'. Typically the user's application is specified here. Example: -b=/bguser/username/hello_world.elf:/bgsys/drivers/ppcfloor/cnk/bgp_kernel.cnk.elf
-lb	Specifies the ELF image to load for I/O nodes. This is typically the vmlinux.elf file. Example: -lb=/bgsys/drivers/ppcfloor/boot/vmlinux-2.6.19
-c	Specifies the path to the directory containing existing core files. Example: -c=/bgusr/username/ You can use the following options: -minicore Specifies the lowest numbered core file to load. The default is 0. Example: -minicore=75 -maxcore Specifies the highest numbered core file to load. The default is 131072. Example: -maxcore=65536
-a	Connects to MMCS server and attaches to the specified, running block. Example: -a=RMP04My010523010 You can use the following options: -host Host name or IP address for the MMCS server. Typically this is either 'localhost' or the Service Nodes' IP address. Example: -host=bgpServiceNode.YourDomain.com -port The TCP port that the MMCS server is listening on for console connections. Example: -port=32031 -user The user name or owner of the mmcs block. Example: -user=bgluser -numbadframes Number of the topmost stack frames to remove from collection. This is useful if the application under debug is doing some customer assembly. Under these conditions the application might not strictly adhere to the PowerPC ABI governing stack frame usage. Example: -numbadframes=1 -numthreads Number of mmcs threads that the Coreprocessor tool will utilize to parallelize data extraction. Example -numthreads=8
-s	Loads the specified Coreprocessor snapshot file for post-failure analysis. Example: -s=/bgusr/username/linpack.snapshot
-templates	Specifies the directory for saved memory templates. Example: -templates=/bgusr/username/mytemplates/.

Option	Description
-nogui	GUI-less mode. Coreprocessor tool processes the data and terminates. Example: -nogui You can use these options: -mode Specifies the type of analysis to perform when Coreprocessor is running in GUI-less mode. Valid modes include: Condensed, Detailed, Survey, Instruction, Kernel, Processor, DCR, PPC, Ungroup_trace, Ungroup, and Neighbor. Example: -mode=Processor -register Specifies the PPC or DCR register when using the PPC or DCR modes. Example: -register=GPR3 -snapshot Fetch failure data and save to snapshot file. Example: -snapshot=/tmp/hang_issue1234
-h	Displays help text. Example: -h

Figure 10-1 shows the Graphical User Interface (GUI) Coreprocessor tool started in a VNC session.

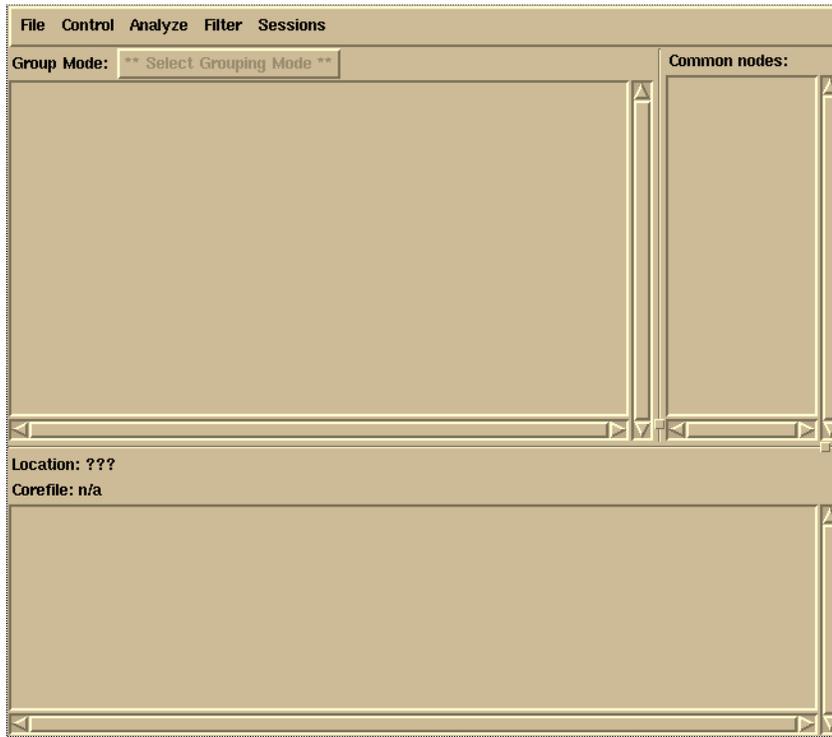


Figure 10-1 Coreprocessor GUI

10.2.2 Debugging live compute node problems

When debugging an unresponsive block, you are looking usually for one or a small subset of compute nodes that are doing something different from the majority of compute nodes.

To do live debug on compute nodes:

1. Start the Coreprocessor GUI as described previously, and then select **File** → **Attach To Block**. The window shown in Figure 10-2 opens.



Figure 10-2 Coreprocessor attach window

2. Supply the following information:
 - Session Name. You can run more than one session at a time, so this is used to distinguish between multiple sessions.
 - Block name.
 - CNK binary (with path). To see both your application and the compute node kernel in the stack, specify the Compute Node Kernel Image and your application binary separated by a colon. For our example, we used:
 - /bgsys/drivers/ppcfloor/boot/cnk:/bgusr/pallas/PMB-MPI1
 - You can specify ION binary as well, but refer to 10.2.4, “Debugging live I/O node problems” on page 158.
 - User name or owner of the MMCS block.
 - TCP port on which MMCS server is listening for console connections (probably 32031).
 - Host name or TCP/IP address for MMCS server. Typically this is *localhost* or the service node's TCP/IP address.
3. Then click **Attach**. At this point, you have not yet affected the state of the processors. Choose **Select Grouping Mode** → **Processor Status**. Notice the text in the upper-left pane (Figure 10-3). The Coreprocessor tool posts the status *?RUN?* because it does not yet know the state of the processors. The number *128* is the number of nodes in the block that are in that state. The number in parenthesis always indicates the number of nodes that share whatever attribute is displayed on the line (the processor state in this case).

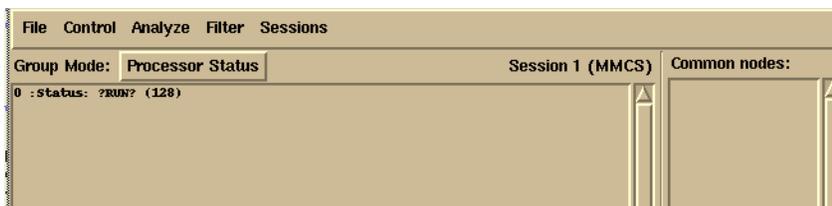


Figure 10-3 Processors in Run status

4. Back at the main screen, click **Select Grouping Mode**. When you choose one of the Stack Traceback options, the Coreprocessor tool halts all the compute node processor cores and displays the requested information. Choose each of the options on that menu in turn so you can see the variety of data formats that are available.

Stack Traceback (condensed)

In the condensed version of Stack Traceback data from all nodes is captured. The unique instruction addresses per stack frame are grouped and displayed, except the last stack frame is grouped based on the function name (not the IAR). This is normally the most useful mode for debug. (See Figure 10-4.)

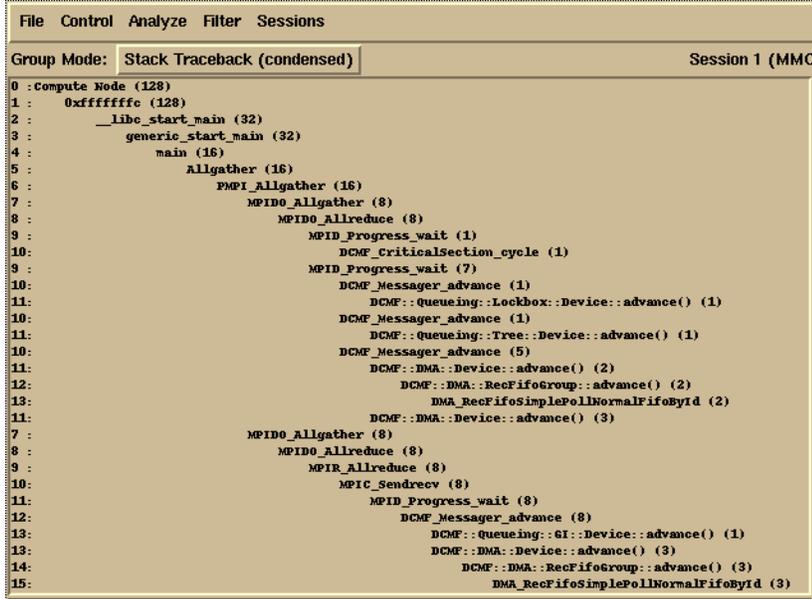


Figure 10-4 Stack Traceback (Condensed)

Stack Traceback (detailed)

In Stack Traceback detailed data from all nodes is captured (see Figure 10-5). The unique instruction addresses per stack frame are grouped and displayed. The IAR at each stack frame is also displayed.

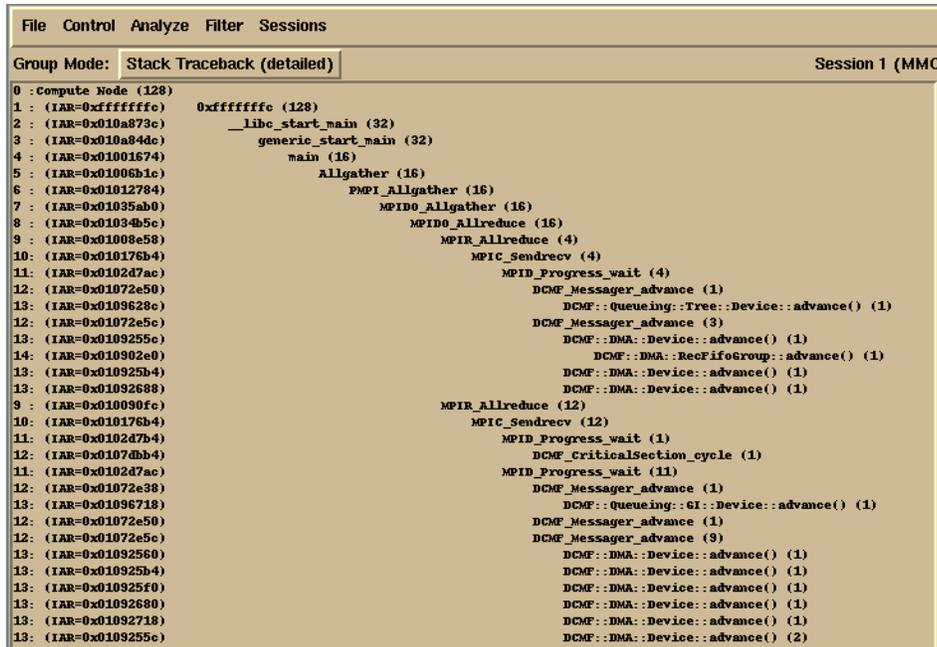


Figure 10-5 Stack Traceback (detailed)

Stack Traceback (survey)

Stack Traceback (survey) is a quick but potentially inaccurate mode. IARs are initially captured and stack data is collected for each node from a group of nodes containing the same IAR. The stack data fetched for that one node is then applied to all nodes with the same IAR. Figure 10-6 shows an example of survey mode.

```

File Control Analyze Filter Sessions
Group Mode: Stack Traceback (survey) Session 1 (MMCS)
0 : Compute Node (128)
1 : 0xfffffff0 (128)
2 :   libe_start_main (32)
3 :     generic_start_main (32)
4 :       main (14)
5 :         MPI_Barrier (14)
6 :           MPID0_Barrier (14)
7 :             MPID0_Progress_wait (14)
8 :               DCMF_Messenger_advance (1)
9 :                 DCMF_Messenger_advance (3)
10:                   DCMF::DMA::Device::advance() (1)
11:                     DCMF::Queueing::Tree::Device::advance() (2)
12:                       DCMF_Messenger_advance (4)
13:                         DCMF::Queueing::GI::Device::advance() (4)
14:                           DCMF_Messenger_advance (6)
15:                             DCMF::DMA::Device::advance() (3)
16:                               DCMF::DMA::RecFifoGroup::advance() (1)
17:                                 DCMF::DMA::RecFifoGroup::advance() (2)
18:                                   DMA_RecFifoSimplePollNormalFifoById (2)
19:                                     DCMF::DMA::Device::advance() (3)
20:                                       main (18)
21:                                         Allgather (18)
22:                                           PMPI_Allgather (18)
23:                                             MPID0_Allgather (18)
24:                                               MPID0_Allreduce (18)
25:                                                 MPIR_Allreduce (2)
26:                                                   MPIC_sendrecv (2)
27:                                                     MPID_Progress_wait (2)
28:                                                       DCMF_Messenger_advance (1)
29:                                                         DCMF::Queueing::Tree::Device::advance() (1)
30:                                                           DCMF_Messenger_advance (1)
31:                                                             DCMF::DMA::Device::advance() (1)
32:                                                               DCMF::DMA::RecFifoGroup::advance() (1)
33:                                                                 MPIR_Allreduce (16)

```

Figure 10-6 Stack Traceback (survey)

The following points can help you use the tool more effectively:

- ▶ The number at the far left, before the colon, indicates the depth within the stack.
- ▶ The number in parentheses at the end of each line indicates how many of the nodes share that same stack frame.
- ▶ If you click any line in the stack dump, the upper-right pane (labeled Common nodes) shows the list of nodes that share that stack frame. See Figure 10-7.

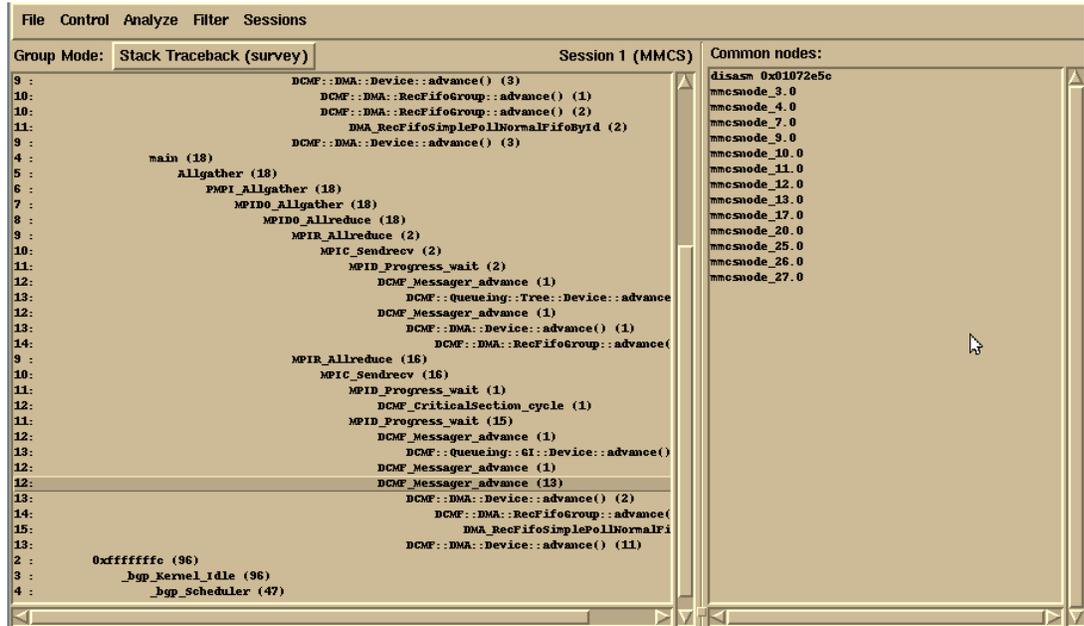


Figure 10-7 Stack Traceback common nodes

- ▶ When you click one of the stack frames and then select **Run** from the Control pull-down, that action is performed for all nodes sharing that stack frame. A new Processor Status summary is displayed. If you chose a Stack Traceback option again, the running processors are halted and the stacks are refetched.
- ▶ You can hold down the Shift key and click several stack frames if you want to control all procedures that are at a range of stack frames.
- ▶ From the Filter pull-down, you can select Create Filter from Group Selection. This adds a filter with the name you specify in the Filter pull-down. When the box for your filter is highlighted, only the data for those processors is displayed in the upper left window. You can create several filters if you want.
- ▶ Set Group Mode to Ungrouped or Ungrouped with Traceback to control one processor at a time.

10.2.3 Saving your information

To save your Traceback information, select **File** → **Save Traceback** to save the current contents of the upper-left pane to a file of your choosing.

To gain more complete data, click **File** → **Take Snapshot™**. Notice that you then have two sessions to choose from on the Sessions pull-down. The original session is MMCS and the second one is SNAP. The snapshot is just what the name implies—a picture of the debug session at a particular point. Notice that you can not start or stop the processors from the snapshot session. You can choose **File** → **Save Snapshot** to save the snapshot to a file. If

you are sending data to IBM for debug, Save Snapshot is a better choice than Save Traceback because the snapshot includes objdump data.

If you choose **File** → **Quit** and there are processors halted, you are given an option to restart them before quitting.

10.2.4 Debugging live I/O node problems

It is possible to debug the I/O nodes as well as compute nodes, but you normally want to avoid doing so. Collecting data causes the processor to be stopped, and stopping the I/O node processors can cause problems with your file system. In addition, the compute nodes will not be able to communicate with the I/O nodes. If you want to do I/O node debug, you must specify the ION binary when you do the **File** → **Attach** to block the pop-up window and choose Debug IONodes from the Filter pull-down menu.

10.2.5 Debugging core files

To work with core files, select **File** → **Load Core**. In the window that opens, specify the following:

- ▶ The location of the CNK binary or binaries
- ▶ The core files location
- ▶ The lowest and highest-numbered core files that you want to work with. (The default is all available core files.)

Click **Load Cores** when you are done.

The same Grouping Modes are available for core file debug as for live debug. Figure 10-8 shows an output example of the Condensed Stack Traceback options from a core file. Condensed mode is the easiest format to work with.

```

File  Control  Analyze  Filter  Sessions
Group Mode: Stack Traceback (condensed)      Session 1 (CORE)  Common nodes:
0 : Compute Node (128)
1 :   0xffffffff (128)
2 :     0x010b350c (32)
3 :       0x010b31cc (32)
4 :         0x010a74a0 (32)
5 :           0x010b37c4 (32)
6 :             0x010dce08 (32)
2 :   0xffffffff (96)
3 :     0x010b37f4 (96)
4 :       0x010b3578 (96)
5 :         _wordcopy_bwd_aligned (1)
6 :           _nl_find_msg (1)
7 :             _nl_load_domain (1)
8 :               __add_to_envIRON (1)
9 :                 vfprintf (1)
10:                   find_module (1)
11:                     __gconv_transliterate (1)
12:                       __quicksort (1)
13:                         __mktime_internal (1)
14:                           0x01085324 (1)
15:                             0x0109b95c (1)
16:                               0x01099fac (1)
17:                                 0x00000000 (1)
5 :   _wordcopy_bwd_dest_aligned (95)
6 :     0x010b92c4 (95)
7 :       0x010dada4 (95)

```

Figure 10-8 Core file condensed stack trace

Figure 10-9 shows the detailed version of the same trace.

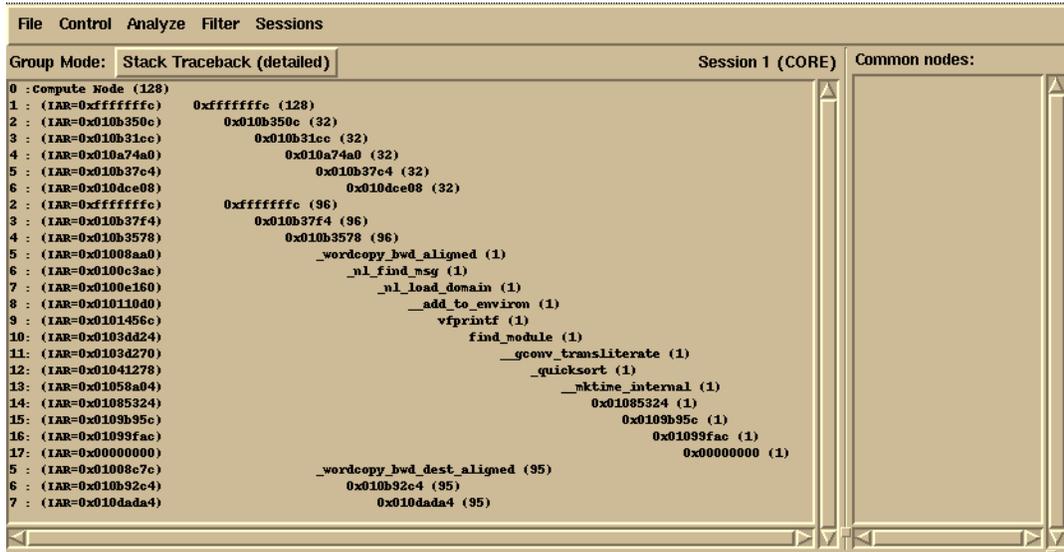


Figure 10-9 Core file detailed stack trace

The Survey option is not as useful for core files because speed is not such a concern.

When you select a stack frame in the traceback output, two additional pieces of information are displayed (Figure 10-10). The core files that share that stack frame are displayed in the Common nodes pane; the Location field under the traceback pane displays the location of that function and the line number represented by the stack frame. If you select one of those core files in the Common nodes pane, the contents of that core file are displayed in the bottom pane.

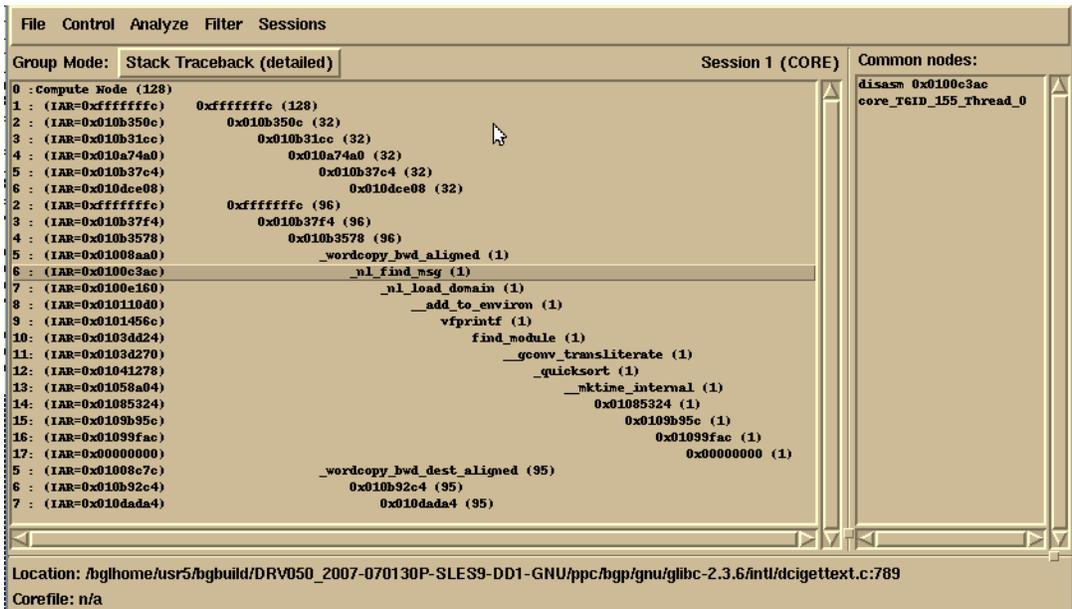


Figure 10-10 Core files common nodes



Configuring the I/O nodes

This chapter provides information about how to configure I/O nodes for your environment.

Note: Configuring GPFS in the Blue Gene/P environment is beyond the scope of this book. For step-by-step instructions, refer to General Parallel File System™ Version 3 Release 2 HOWTO for the IBM System Blue Gene/P Solution, which provides Blue Gene-specific information for installing and configuring GPFS. To view the latest version, go to the IBM Publications Center at the following address, select your country, and search on publication number SC23-5230:

<http://www.elink.ibm.link.ibm.com/public/applications/publications/cgibin/pbi.cgi>

11.1 Using the I/O node startup and shutdown scripts

The I/O node configuration centers around startup and shutdown, which is always done in a predefined sequence. To understand how to configure the I/O nodes, you must know which scripts and files are used and how and when they interact.

11.1.1 BusyBox

BusyBox provides the shell for the I/O node. It is a small specialized shell that enables a subset of commands and command options. The I/O node boot scripts run the BusyBox commands. For more information about BusyBox, refer to the following Web address:

<http://www.busybox.net/>

11.1.2 Site customization directory and the ramdisk

The ramdisk contains the root file system that is needed by the I/O node kernel. Table 11-1 lists the directories and files that are important for I/O nodes.

Table 11-1 Ramdisk contents

Directory or file	Description
/bgsys	Network File System (NFS) mount point of /bgsys
/bin	NFS mount point of /bgsys/drivers/ppcfloor/linux/OS/bin
/bin.rd	BusyBox and Blue Gene programs
/ciod.rd	Internal working directory for CIOD
/etc/ntp.conf	NTPD configuration file
/etc/rc.status	Common functions for rc scripts
/etc/sysconfig/ciod	CIOD configuration file
/etc/sysconfig/gpfs	GPFS configuration file
/etc/syslog.conf	Syslogd configuration file
/proc/personality	Binary personality file
/proc/personality.sh	Script with personality info that can be sourced by shell scripts
/sbin	NFS mount point of /bgsys/drivers/ppcfloor/linux/OS/sbin
/sbin.rd	BusyBox and Blue Gene programs

11.1.3 Startup flow

The I/O node kernel boots at first from the ramdisk and then switches to a combination of ramdisk content and NFS-mounted content later in the boot. The kernel transfers control to /sbin/init after the kernel completes initialization. A simple inittab file is provided in the ramdisk that gives control to /etc/init.d/rc.sysinit.

Here is an overview of the operations performed by the script rc.sysinit:

1. Mount / and /proc file systems.
2. Source /proc/personality.sh to set variables to personality values.
3. Source /etc/sysconfig/sysinit for site-specific I/O node initialization settings.

4. Validate the MAC address.
5. Initialize the functional network interface (eth0).
6. Initialize the local network interface (lo).
7. Add route to network gateway.
8. Mount /bgsys.
9. Copy all files/directories from /bgsys/iofs to /.
10. Set the host name from /etc/hosts if it exists, otherwise use the IP from the personality (\$BG_IP).

Note: For steps 11, 12, 13, and 14, you can configure the path /bgsys/drivers/ppcfloor/linux/OS/bin to point elsewhere. You can also set the path using the environment variable \$BG_OSDIR.

11. Rename /bin to /bin.rd and make a symbolic link from /bin to /bgsys/drivers/ppcfloor/linux/OS/bin.
12. Rename /sbin to /sbin.rd and make a symbolic link from /sbin to /bgsys/drivers/ppcfloor/linux/OS/sbin.
13. Make symbolic link from /usr to /bgsys/drivers/ppcfloor/linux/OS/usr.
14. Make symbolic links from /lib to any shared libraries in /bgsys/drivers/ppcfloor/linux/OS/lib that do not already exist in /lib.
15. Run ldconfig to create the shared library cache.
16. Run any scripts of the form /etc/init.d/rc3.d/S[0-9][0-9]*.
17. Signal the control system that the I/O node has completed boot.

11.1.4 Shutdown flow

When an I/O node is halted the shutdown rule in /etc/inittab is run. This rule executes the script /etc/rc.shutdown. The shutdown flow is basically the reverse of the startup flow although it is less complicated. The shutdown flow is as follows:

1. Run any scripts of the form /etc/init.d/rc3.d/K[0-9][0-9]*.
2. Flush the filesystem buffers.
3. Unmount all file systems.
4. Call the BusyBox implementation of halt.

11.1.5 Shell variables from personality info

The special file /proc/personality.sh is a shell script that sets shell variables to node-specific values. See Table 11-2. These are used by the rc.* scripts and might also be useful within site-written start and shutdown scripts. These are not exported variables, so each script must invoke this file with the **source** command or export the variables to other scripts.

Table 11-2 Shell variables available during I/O node startup and shutdown

Shell variable	Description
BG_BLOCKID	Name of the block
BG_BROADCAST	Broadcast address for this I/O node
BG_CLOCKHZ	Processor frequency in megahertz

Shell variable	Description
BG_EXPORTDIR	File server export directory for /bgsys
BG_FS	File server IP address for /bgsys
BG_GATEWAY	Gateway address for the default route for this I/O node
BG_GLINTS	Non-zero value when global interrupts are enabled
BG_IP	IP address of this I/O node
BG_ISTORUS	String to indicate if torus HW is present (normally empty string on I/O nodes).
BG_LOCATION	Location string for this I/O node, of the form Rrr-Mm-Nnn-Jjj where rr is the rack number, m is the midplane number, nn is the node number, and jj is the slot number. for example, R04-M1-N01-J01 Use this variable rather than hard coding the I/O node location names.
BG_MAC	Ethernet MAC address of this I/O node
BG_MTU	MTU for this I/O node
BG_NETMASK	Netmask to be used for this I/O node
BG_NODESINPSET	Number of compute nodes served by this I/O node
BG_NUMPSETS	Total number of psets in this block (that is, the number of I/O nodes)
BG_PSETNUM	Processor set (pset) number for this I/O node (0..\$BG_NUMPSETS-1)
BG_PSETORG	Shows the X, Y, Z coordinates for nodes
BG_PSETSIZE	String with three numbers, size of X dimension, size of Y dimension, and size of Z dimension
BG_SIMULATION	Non-zero if block is running under simulation
BG_SNIP	Functional network IP address of the Service Node
BG_UCI	Universal component ID for this I/O node
BG_VERBOSE	Non-zero if the block is created with the <i>io_node_verbose</i> option"
BG_XSIZE, BG_YSIZE, BG_ZSIZE	Size of block in nodes, one variable for each of the three dimensions

11.1.6 I/O node log file

Output written to the console is sent through the mailbox to the control system and stored in a log file in the /bgsys/logs/BGP directory. The log file name is the I/O node's location string with *.log* appended to the end. For example, R01-M0-N04-J01.log. Output from the kernel, startup and shutdown scripts, and daemons is put in the log file and can be helpful for diagnosing problems or errors.

11.2 Configuring I/O node services

This section addresses the configuration of I/O node services including the Control and I/O Daemon (CIOD), the Network Time Protocol (NTP) daemon, Syslog, and the Network File System (NFS).

11.2.1 Configuring CIO daemon

The Control and I/O Daemon (CIOD) is started by the `/etc/init.d/rc3.d/S50ciod` script. The configuration information for CIOD is stored in the `/etc/sysconfig/ciod` file. Table 11-3 lists the environment variables used by CIOD. You can change the default options for CIOD by providing a `/bgsys/ios/etc/sysconfig/ciod` file. For example, to change the default read/write buffer size to 512 KB, the `/etc/sysconfig/ciod` file should contain the following line:

```
export CIOD_RDWR_BUFFER_SIZE=524288
```

Environment variables used by CIOD

Table 11-3 describes the environment variables used by CIOD and when they were implemented.

Table 11-3 Environment variables used by CIOD

Environment variable	Description
CIOD_ENABLE_CONTROL_TRACE	When the value is non-zero, CIOD turns on a control trace when it starts. The default value is 0.
CIOD_END_JOB_TIMEOUT	The value is the number of seconds CIOD waits before it assumes the ioproxy process is hung when the job is ended. When the timeout expires, CIOD ends the ioproxy process by sending a SIGKILL signal. The default value is 4.
CIOD_IOPROXY_AFFINITY_MASK	The value is a bit mask that controls what cores the ioproxy processes are allowed to run on. For example, to allow the ioproxy processes to run on cores 0, 1, and 2, the value is 7. The default value is 15 to allow the ioproxy processes to run all cores.
CIOD_LOG_LEVEL	The value controls the amount of data written by CIOD to the I/O node log file. When the value is 0, CIOD does no additional logging. When the value is 1, CIOD logs additional information for recoverable errors. When the value is 2, CIOD also logs when jobs start and end. When the value is 3, CIOD also logs performance data for selected system calls. The log level is accumulative so that when the value is 3 the information for log levels 1 and 2 is also written to the log file. The default value is 0.
CIOD_MAIN_AFFINITY_MASK	The value is a bit mask that controls what cores the main CIOD threads are allowed to run on. For example, to allow the main threads to run on cores 0 and 1, the value is 3. The default value is 15 to allow the main threads to run all cores.
CIOD_MAX_CONCURRENT_LOAD_OPS	The value is the maximum number of concurrent program load operations from this I/O node. A value of 0 means there is no limit to the number of concurrent load operations. Note that with a limit it is possible for blocking operations to keep this I/O node from making progress. The default value is 4 when the block is running in HTC mode. Otherwise the default value is 0.
CIOD_MAX_CONCURRENT_RDWR_OPS	The value is the maximum number of concurrent read and write operations from this I/O node. A value of 0 means there is no limit to the number of concurrent operations. Note that with a limit it is possible for blocking operations to keep this I/O node from making progress. The default value is 0.

Environment variable	Description
CIOD_NEW_JOB_EXIT_PGM	The value is the path to a program that is called each time a new job is started. The program is passed eight parameters describing the new job (see “CIOD new job program” on page 166 for details). If the program returns a non-zero exit status, CIOD returns an error to the control system and the job does not start. The default value is null.
CIOD_NEW_JOB_EXIT_PGM_TIMEOUT	The value is the number of seconds that CIOD waits before it assumes the new job exit program is hung. When the timeout expires, CIOD ends the program, returns an error to the control system and the job does not start. When the value is -1, CIOD waits forever. The default value is -1.
CIOD_READER_AFFINITY_MASK	The value is a bit mask that controls what core the collective network reader thread is allowed to run on. For example, to allow the reader thread to run on core 3, the value is 8. The default value is 15 to allow the reader thread to run all cores.
CIOD_READER_INTERRUPT_MODE	When the value is non-zero, the collective network reader thread waits to be interrupted when a packet is available in the collective network reception fifo. When the value is zero, the reader thread polls the collective network reception fifo looking for packets. The default value is 0.
CIOD_RDWR_BUFFER_SIZE	This value specifies the size, in bytes, of each buffer used by CIOD to issue read and write system calls. One buffer is allocated for each compute thread associated with this I/O node. The default size, if not specified, is 262144 bytes. A larger size might improve performance. If you are using a GPFS file server, we recommend that you ensure that this buffer size matches the GPFS block size. Experiment with different sizes, such as 262144 (256 K) or 524288 (512 K), to achieve the best performance.
CIOD_REPLY_MSG_SIZE	The value is the size in bytes for reply messages sent from CIOD to the compute nodes. The value cannot be larger than the value of the CIOD_RDWR_BUFFER_SIZE variable. A smaller value causes CIOD to return the data from a system call in multiple messages and spend shorter intervals injecting packets into the collective network device. The default value is 262144.
CIOD_TRACE_DIRECTORY	The value is the path to the directory CIOD uses to store trace files. The default value is NULL.
CIOD_TREE_RECV_RETRY	The value is the number of times that CIOD tries to receive a packet from the collective network device before it decides that a packet is not available and posts ready messages. A larger value causes CIOD to pay more attention to the collective network. The default value is 50.

CIOD new job program

CIOD can call a program provided by the administrator each time a new job starts. For example, the program can be used to tune the file system client on a per job basis or mount specific file systems based on the user.

The program is specified by setting the CIOD_NEW_JOB_EXIT_PGM environment variable in the `/etc/sysconfig/ciod` file. The program runs as root in a new process group. This allows the program to perform the same functions as an rc script that runs at startup or shutdown. When the program runs, all of the compute nodes have loaded the application and are waiting to start the application.

A timeout can be specified by setting the CIOD_NEW_JOB_EXIT_PGM_TIMEOUT environment variable in the `/etc/sysconfig/ciod` file. If the program does not complete in the

number of seconds specified by the timeout, CIOD ends the process group by sending a SIGKILL signal. The default value of -1 causes CIOD to wait forever.

If there is an error in starting the program, CIOD allows the job to continue. If the program returns a non-zero exit status or waiting for the program to complete times out, CIOD sends an error reply back to the control system. The control system then marks the job as *failed* and CIOD resets and waits for the next job.

The program is passed the following parameters:

- ▶ Indicator of job mode (“0” for coprocessor mode, “1” for virtual node mode or “2” for dual mode)
- ▶ Number of physical compute nodes managed by CIOD on this I/O node
- ▶ Number of compute nodes processes that are participating in the job
- ▶ User ID of current user
- ▶ Group ID of current user
- ▶ Job number of the job
- ▶ Number of jobs run so far on this boot
- ▶ Path to program running on compute nodes

Service Connection

CIOD supports a service connection that allows the administrator to check on a running CIOD to get status or enable and disable a trace facility. The service connection is accessed by using telnet to port 9000 on the I/O node. The administrator enters commands to get overall status of CIOD and the compute nodes it is managing. In addition, the administrator can dynamically enable and disable a trace facility that causes CIOD to log control flow.

11.2.2 Configuring Network Time Protocol

The Network Time Protocol daemon (NTPD) is started by the `/etc/init.d/rc3.d/S05ntp` script. The configuration information for the NTPD is stored in the `/etc/ntp.conf` file. If the file does not exist, a default file is created by the `S05ntp` script, which includes the following lines:

```
restrict default nomodify
server $BG_SNIP
```

You can change the default options for NTPD by providing a `/bgsys/iofs/etc/ntp.conf` file. For example, you can change the NTP server to something other than the Service Node.

You must also set your time zone file. The I/O node looks for the time zone in the file `/etc/localtime`. To set up your time zone file, copy your Service Node's `/etc/localtime` file to `/bgsys/iofs/etc/localtime`. If the time zone file is not found, the time is assumed to be UTC.

For more information about NTP, refer to the following Web address:

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

11.2.3 Configuring the Syslog daemon

The `syslogd` and `klogd` daemons are started by the `/etc/init.d/rc3.d/S02syslog` script. The default system log file name is the I/O node's location string with `.log` appended. For example:

```
R01-M1-N12-J01.log
```

The configuration information for `syslogd` is stored in the `/etc/syslog.conf` file. Example 11-1 shows the contents of the default file that is included with the ramdisk:

Example 11-1 Contents of syslog.conf

```

# /etc/syslog.conf - Configuration file for syslogd(8)
# For info about the format of this file, see "man syslog.conf".
#
# print most on tty10 and on the xconsole pipe
#
kern.warning;*.err;authpriv.none /dev/tty10
kern.warning;*.err;authpriv.none|/dev/xconsole
*.emerg          *

# enable this, if you want that root is informed
# immediately, e.g. of logins
#*.alert        root
#
# Warnings in one file
#
*.=warning;*.=err-/var/log/warn
*.crit          /var/log/warn
#
# save the rest in one file
#
*.*;mail.none;news.none-/var/log/messages
#
# enable this, if you want to keep all messages
# in one file
#*.*            -/var/log/allmessages
#
# Some foreign boot scripts require local7
#
local0,local1.*-/var/log/localmessages
local2,local3.*-/var/log/localmessages
local4,local5.*-/var/log/localmessages
local6,local7.*-/var/log/localmessages

```

You can change the default options for syslogd and klogd by providing a `/bgsys/iofs/etc/syslog.conf` file. The log files can get quite large. Consider using a utility to manage these files, such as logrotate.

For more information, refer to the man pages for syslogd, klogd, and logrotate.

11.3 I/O node performance tips

Two areas in which you can improve your I/O performance on Blue Gene/P are CIOD buffer size and your NFS mount options. This section contains some areas where you might try tweaking them to enhance your I/O.

11.3.1 Configuring the CIOD buffer size

I/O performance might be improved by increasing the size of the CIOD buffers used to issue read and write system calls. One CIOD buffer is allocated in the I/O node per compute node thread in the processor set (pset). The buffer size (in bytes) can be changed by creating a `/bgsys/iofs/etc/sysconfig/ciod` file to set the `CIOD_RDWR_BUFFER_SIZE` environment variable. See 11.2.1, “Configuring CIO daemon” on page 165 for details on configuring CIOD.

Note: The buffer size value is per compute node thread. For example, if the CIOD read/write buffer size is set to 256 K and there are 32 compute nodes per processor set with each compute node having four threads, then CIOD allocates a total of 32768 K for read/write buffers.

11.3.2 Configuring Network File System

Blue Gene/P currently supports only Network File System (NFS) version 3 (V3). To set up NFS for Blue Gene/P, follow these steps:

1. Configure the number of threads that the NFS server will use in the `/etc/sysconfig/nfs` file. The default for the `USE_KERNEL_NFSD_NUMBER` is four. You can adjust that number to suit your environment.
2. Next, export the file system. Edit the `/etc/exports` file, and add the following line (assuming the I/O nodes will be in the 172.16.x.x network):

```
/bgsys 172.16.0.0/255.255.0.0
```

Then export using the following command:

```
exportfs -ar
```

3. Now you will be able to mount `/bgsys` from a client using the following command:

```
mount -o soft 172.16.1.1:/bgsys /bgsys
```

11.3.3 Performance tips for the NFS file server

Consider the following points when you perform the configuration on your NFS file server:

- ▶ Increase the number of NFS server daemons (`/etc/sysconfig/nfs`, `USE_KERNEL_NFSD_NUMBER="xxx"`). Try using 32, 64, or 128.
- ▶ If you use the `noac` (no attribute cache) option when mounting NFS directories to avoid problems with multiple users writing to the same file, you might experience slower performance. Administrators should consider mounting different file systems with different attributes to achieve the best performance.
- ▶ On a Linux server, use the `sysctl` command to change the settings that pertain to TCP and sockets.

Note: The following values might reflect the extreme. The values you supply will depend on your system and memory.

- This setting turns off TCP timestamp support; the default is on.

```
net.ipv4.tcp_timestamps = 0
```

- This sets the TCP time-wait buckets pool size; the default is 180000.

```
net.ipv4.tcp_max_tw_buckets =2000000
```

- This sets the min/default/max TCP read buffer; the defaults are 4096, 87380, and 174760.

```
net.ipv4.tcp_rmem =10000000 10000000 10000000
```

- This sets the min/pressure/max TCP write buffer; the defaults are 4096, 16384, and 131072.

```
net.ipv4.tcp_wmem =10000000 10000000 10000000
```

- This sets the min/pressure/max TCP buffer space; the defaults are 31744, 32256, and 32768.
`net.ipv4.tcp_mem =10000000 10000000 10000000`
 - This turns off SACK support; the default is on.
`net.ipv4.tcp_sack =0`
 - This turns off TCP window scaling support; the default is on.
`net.ipv4.tcp_window_scaling =0`
 - This is the maximum number of skb-heads to be cached; the default is 128.
`net.core.hot_list_length =20000`
 - This is the maximum receive socket buffer size; the default is 131071.
`net.core.rmem_max =10000000`
 - This is the maximum send socket buffer size; the default is 131071.
`net.core.wmem_max =10000000`
 - This is the default receive socket buffer size; the default is 65535.
`net.core.rmem_default =10000000`
 - This is the default send socket buffer size; the default is 65535.
`net.core.wmem_default =10000000`
 - This is the maximum amount of option memory buffers; the default is 10240.
`net.core.optmem_max =10000000`
`net.core.netdev_max_backlog =300000`
- ▶ On the NFS mount command, described in Table 11-4, specifying larger values for *rsize* and *wsize*, using *async*, and using *tcp* might improve performance.

11.4 Typical site customization

A Blue Gene/P site normally has a script that customizes the startup and shutdown. This script can perform any or all of the following functions in the order shown:

1. Mount (during startup) and unmount (during shutdown) high performance file servers for application use. To ensure that the file servers are available to applications:
 - The mount must occur after portmap is started but before CIOD is started by the S50ciod script.
 - The unmount must occur after CIOD is ended by the K50ciod script and before portmap is ended.

For NFS mounts, we recommend that you:

- Retry the mount several times to accommodate a busy file server.
- See Table 11-4 for the recommended mount options.

Table 11-4 Recommended mount options for NFS

Option	Discussion
tcp	This option provides automatic flow control when it detects that packets are being dropped due to network congestion or server overload.

Option	Discussion
rsizewsize	These options specify read and write NFS buffer sizes. The minimum recommended size is 8192, and the maximum recommended size is 32768. If your server is not built with a kernel compiled with a 32768 size, it negotiates down to what it can support. In general, the larger the size is, the better the performance is. However, depending on the capacity of your network and server, a size of 32768 might be too large and cause excessive slowdowns or hangs during times of heavy I/O. Each site needs to tune this value.
async	Specifying this option might improve write performance, although there is greater risk for losing data if the file server crashes and is unable to get the data written to disk.

2. Set the CIOD environment variables. See 11.2, "Configuring I/O node services" on page 165.
3. Set NTP parameters. See 11.2, "Configuring I/O node services" on page 165.
4. Set syslog parameters. See 11.2.3, "Configuring the Syslog daemon" on page 167.

Your site customization script should be in the `/bgsys/iofs/etc/init.d` directory. For example, call it `sitefs`. Then to properly place it in the startup and shutdown sequence, symbolic links should be created to this script as follows:

```
ln -s /bgsys/iofs/etc/init.d/sitefs /bgsys/iofs/etc/init.d/rc3.d/S10sitefs
ln -s /bgsys/iofs/etc/init.d/sitefs /bgsys/iofs/etc/init.d/rc3.d/K90sitefs
```

The `S10sitefs` and `K90sitefs` links are sequenced so that they meet the mount requirements described previously:

- ▶ During startup, `S10sitefs` is called with the `start` parameter.
- ▶ During shutdown, `K90sitefs` is called with the `stop` parameter.

Example 11-2 illustrates a site customization script that mounts a user file system at `/bgusr`. An application called `bgras` is used to generate a RAS event if `/bgusr` cannot be mounted.

Example 11-2 Sample site customization script

```
#!/bin/sh
#
# This will run after portmap so NFS can be mounted with locking.
# It runs before CIOD so the mounted filesystems will be available for the
# first job that might run.
#
# Redirect to the console.
exec > /dev/console 2>&1

. /etc/rc.status

SITEFS=172.16.1.101

rc_reset

case "$1" in
    start)
        echo "Mounting site filesystems"
        [ -d /bgusr ] || mkdir /bgusr
```

```

let ATTEMPT=1
let ATTEMPT_LIMIT=5
NFS_OPTS=mountvers=3,rsz=8192,wsz=8192,tcp,async,lock,noac
until test $ATTEMPT -gt $ATTEMPT_LIMIT || mount $SITEFS:/bgusr /bgusr -o
$NFS_OPTS; do
    sleep $ATTEMPT
    let ATTEMPT=$ATTEMPT+1
done
if test $ATTEMPT -gt $ATTEMPT_LIMIT; then
    echo "All attempts to mount $SITEFS:/bgusr failed... giving up."
    bgras 1 34 2 "Mount of $SITEFS:/bgusr failed."
fi

rc_status -v
;;

stop)
echo "Unmounting site filesystems"
umount /bgusr -l

rc_status -v
;;

restart)
echo "Restart not implemented."
;;

status)
echo "Status not implemented."
;;

esac
rc_exit

```

A more advanced script, such as the one shown in Example 11-3, can select a file server based on the I/O node location string, for example, R01-M0-N12-J02. Note that `BG_LOCATION` comes from invoking `/proc/personality.sh` with the `source` command.

Example 11-3 Selecting file server based on I/O node location string

```

case "$BG_LOCATION" in
R00-*) SITEFS=172.32.1.1;;# rack 0 NFS server
R01-*) SITEFS=172.32.1.2;;# rack 1 NFS server
R02-*) SITEFS=172.32.1.3;;# rack 2 NFS server
R03-*) SITEFS=172.32.1.4;;# rack 3 NFS server
R04-*) SITEFS=172.32.1.5;;# rack 4 NFS server
R05-*) SITEFS=172.32.1.6;;# rack 5 NFS server
esac

```

The script can use `$BG_IP` (the IP address of the I/O node) to compute the file server or servers to select. It can also use `$BG_BLOCKID` so that different blocks can mount special file servers.

11.4.1 Additional GPFS notes

The GPFS file system is not enabled by default. To enable GPFS daemons and the co-requisite SSH daemon, you must first create the files `/bgsys/iofs/etc/sysconfig/gpfs` and `/bgsys/iofs/etc/sysconfig/ssh`. The file `/bgsys/iofs/etc/sysconfig/gpfs` must set the variable `GPFS_STARTUP` to 1. The file `/bgsys/iofs/etc/sysconfig/ssh` must declare the variable `SSHD_OPTS`. Unless specific SSH options are desired, `SSHD_OPTS` can be set to an empty string. For more information about GPFS, refer to the HOWTO mentioned at the start of this chapter.

If you have customized your startup and shutdown routines, make sure that you have included the following section to insure that GPFS and SSH daemons start:

```
echo "GPFS_STARTUP=1" >> /etc/sysconfig/gpfs
  if [ ! -r /etc/sysconfig/ssh ]; then
    echo "SSHD_OPTS=\"\" > /etc/sysconfig/ssh
  fi
```

To verify that `sshd` is running on the I/O nodes, follow these steps:

1. From the MMCS console, allocate a block.
2. Using the `{i}` locate command (from the MMCS console) determine the locations of your I/O nodes.
3. Verify that the daemon is running by using the `write_con` command:

```
{i}write_con /etc/init.d/sshd status
```

If the daemon is running all is OK.

If `sshd` is not running on the I/O nodes, then complete the following:

1. Verify that the `/bgsys/iofs/etc/ssh/sshd_config` file exists. If not you can copy the file from the `/etc/ssh` directory. The permissions should be set to 755.
2. Verify the following in the `sshd_config` file:


```
# UsePAM yes - #(Ensure this line is commented out)
PermitRootLogin yes
```
3. Verify that you have the `/bgsys/iofs/etc/sysconfig` directory. Permissions should be 755.
4. Verify that you have the file `/bgsys/iofs/etc/sysconfig/ssh`. If it doesn't exist then you can copy it from the `/bgsys/linux/xxxxxxx/etc/sysconfig` directory.
5. Use the `write_con` command on the I/O node to start the `sshd` (or reboot the I/O node, it should start automatically).

```
{i}write_con /etc/init.d/sshd start.
```

Try to SSH to the I/O node, either by host name or IP address. If you still cannot connect check the log files for errors.

If you are going to allow users to log into the I/O nodes then it is a good idea to copy `/etc/passwd`, `/etc/group` and `/etc/shadow` into the `/bgsys/iofs/etc` directory. That way the users can log into the I/O nodes as themselves.



Power management

The Blue Gene/P racks are capable of controlling power consumption that is used by applications through an automatic process of each rack and also through a power control process that is initiated through the job submission features that are supported in Blue Gene/P. These features allow the power to be controlled on each job submitted, regardless of the block size used. This chapter describes the power management features of Blue Gene/P.

12.1 Blue Gene/P power management

The goal of power management is to allow management of power that is used while applications are running. It also enables a repeatable set of performance results during power managed application runs. In addition, this approach allows user level power utilization control of the system. There are two layers of power management: kernel and DC/DC converter. *Kernel level* power management uses available performance counters to indicate that a management event is required. *DC/DC converter type* management polls status registers in the DC/DC voltage converters regular intervals. Consecutive readings in excess of the maximum settings will trigger a power management event.

12.2 Invoking power management

Power management can be invoked in one of two ways: *reactively* or *pro-actively*. Both methods achieve the same goal—they reduce the power consumption by the application. The advantage to using proactive power management is that you are better able to control the power utilized during the entire application run.

12.2.1 Reactive power management

Reactive power management is initiated at the DC/DC voltage converter. A status register that is contained in the converter is polled at regular intervals. If the amount of current exceeds the threshold the converters begin managing the power. Initially the power is reduced by 10%. The application continues to run; however, if it reaches the threshold again, the power is reduced another 10%.

12.2.2 Proactive power management

If you planning on running a job that you know will draw too much power, you can invoke proactive power management when submitting the job. Proactive power management is enabled on a per job basis. Proactive power management uses the same controls as the reactive, but the controls are specific to the job. The settings invoked for the job will end when the job has terminated.

Currently, there are three environment variables that can be appended to a submit_job to invoke proactive power management or set as environment variables:

- ▶ BG_POWERMGMTDUR number of microseconds to spend in idle loop
- ▶ BG_POWERMGMTPERIOD Proactive throttling period (in microseconds)
- ▶ BG_POWERMGMTDDRRATIO DDR throttling ratio %

Here is an example of submitting a job (from MMCS console) invoking power management:

```
submit_job <job> <outfile> <mode> BG_POWERMGMTDUR=100 BG_POWERMGMTPERIOD=1000  
BG_POWERMGMTDDRRATIO=10
```

In this example, every 1000 microseconds all nodes in the system take an interrupt, sit in a while loop for 100 microseconds and then return (100/1000=10%).

The POWERMGMTDUR and POWERMGMTPERIOD must be used together. POWERMGMTDDRRATIO can be combined with the other two or can be used alone.

12.3 RAS messages

When power management occurs, either reactive or proactive, there are RAS messages generated. When the threshold has been exceeded all nodes in the block enter power management mode.

- ▶ Reactive power management causes the following message:
KERN_180D Severity: Warning Message Text: Environment Monitor. Partition now in reactive power management mode
- ▶ Each node card that exceeds the max power threshold generates the following message:
KERN_1809 Severity: Warning Message Text: Environment Monitor threshold warning (Max Power). Status=0x00002000 PTMON Warnings=0x0x00000000, Maximum Temperature=26 C, 1.2v Domain power=415 watts, 1.8v Domain power=265 watts, TotalPower=670 watts EnvError=0x00000000
- ▶ If proactive power management is invoked, you see the following message:
KERN_180E Severity: Warning Message Text: Environment Monitor. Partition now in proactive power management mode

In addition to the RAS messages that are generated, you can see current jobs that are running with power management invoked in the Navigator. On the Health Center page, these jobs are listed under one of two categories:

- ▶ Running Jobs with Proactive Power Management
- ▶ Running Jobs with Reactive Power Management



Cluster Systems Management for Blue Gene/P

This chapter provides a high-level introduction to Cluster Systems Management (CSM) and its use with Blue Gene/P. CSM support for Blue Gene/P was made available with CSM 1.7. For the detailed information that you need to plan, install, and run CSM, see the CSM product documentation.

13.1 Overview

The Blue Gene/P control system software records configuration, RAS, and environmental information in the Blue Gene/P DB2 database. It also provides a Web interface and several command line interface (CLI) tools for working with the information that is stored in the database. However, it is up to the Blue Gene administrators and users to watch or check the database, determine when the database entries are indicative of a problem, and then take appropriate action.

CSM is an IBM licensed software product that is used to manage clusters of AIX® and Linux systems, from one to thousands. CSM has many capabilities, but in this chapter we focus on just one: its rich event monitoring and automated response capability. With CSM, you can specify what constitutes an event, and what should happen, automatically, if and when the event occurs.

By installing CSM on your Blue Gene Service Node, you can automate the task of watching the database and taking corrective actions. For example, you could use CSM to watch the status of all the midplanes. If a midplane is marked in error, or is marked as missing, the CSM software can detect this and take whatever action you've specified, automatically. Perhaps you want to be paged, or receive an urgent e-mail, a special script should be run. Or perhaps all of these responses should happen. CSM allows you to set up whatever monitoring and automated responses you need.

To use CSM with your Blue Gene/P in the simplest manner possible, begin with a fully installed, fully operational Blue Gene, including Service node, Front End nodes, and File Servers. Next, obtain CSM through your IBM sales representative or the full-featured 60 day try-and-buy version from:

<http://www14.software.ibm.com/webapp/set2/sas/f/csm/home.html>

Follow the instructions in the CSM Planning and Installation Guide to install and configure the CSM management server software on your Service Node. Then, follow the instructions in the same book for adding the optional CSM support for Blue Gene.

In the sections that follow, we discuss CSM's monitoring and automated response capabilities, and how to use them. Additional information can be found in the following CSM and Reliable Scalable Cluster Technology (RSCT) product publications (all available at the link mentioned above):

- ▶ CSM Administration Guide
- ▶ CSM Command and Technical Reference
- ▶ RSCT Administration Guide
- ▶ RSCT Technical Reference

13.2 Monitoring the Blue Gene/P database with CSM

When CSM (and the optional Blue Gene/P support) is installed and configured on your Service node, you can monitor the Blue Gene database for events of interest using a few simple commands.

First, associate a condition with a response by running the **mkcondresp** command, for example:

```
mkcondresp BGNodeErr BroadcastEventsAnyTime
```

Then, start monitoring that condition by running the following command:

```
startcondresp BGNodeErr
```

A condition is a persistent CSM monitoring construct that identifies what to monitor, and what to monitor for. In this example, BGNodeErr is a condition. Basically, BGNodeErr is concerned with the Blue Gene database table TBGPNode, and in particular, is concerned with TBGPNode row updates that set the status column to *E* or *M*.

A response is a persistent CSM monitoring construct that identifies an action to take. In this example, BroadcastEventsAnyTime is a response that puts up a wall message for each event passed to it.

An event is a dynamic CSM monitoring construct generated by CSM when a monitored condition's event expression evaluates true (i.e. when that which is being monitoring for occurs).

By running **startcondresp BGNodeErr** you are effectively telling CSM to monitor the Blue Gene database table TBGPNode for row updates that set the status column to *E* or *M*. This means that CSM is expected to generate an event whenever either type of row update occurs. Because you ran **mkcondresp BGNodeErr BroadcastEventsAnyTime** before that, CSM also knows that it should pass all such events to the **BroadcastEventsAnyTime** response, which, by design, puts up a wall message for each event passed to it.

BGNodeErr is a *predefined* condition. CSM provides many predefined conditions. To get a list, simply run **lscondition**. To learn what a particular condition is for, run **lscondition condition_name**, as shown in Example 13-1.

Example 13-1 The lscondition command

```
# lscondition BGNodeErr
Displaying condition information:

condition 1:
  Name           = "BGNodeErr"
  Node           = "bgpdd1sys1"
  MonitorStatus  = "Not monitored"
  ResourceClass  = "IBM.Sensor"
  EventExpression = "SD.Uint32>0 && SD.Int32==1"
  EventDescription = "An event will be generated when the status of an I/O
or Compute Node is marked \"E\" (for error) or \"M\" (for missing).\"
  RearmExpression = "SD.Uint32>0 && SD.Int32==-1"
  RearmDescription = "A rearm event will be generated when the status of the
same I/O or Compute Node is marked \"A\" (for available) again.\"
  SelectionString = "Name==\"BGNodeErr\""
  Severity       = "c"
  NodeNames      = {}
  MgtScope       = "1"
  Toggle         = "No"
```

BroadcastEventsAnyTime is a predefined response. CSM provides many predefined responses. To get a list, run **lsresponse**, as shown in Example 13-2. To learn what a particular response does, run **lsresponse response_name**.

Example 13-2 The lsresponse command

```
# lsresponse BroadcastEventsAnyTime
Displaying response information:
```

```

ResponseName    = "BroadcastEventsAnyTime"
Node            = "bgpdd1sys1"
Action          = "wallEvent"
DaysOfWeek      = 1-7
TimeOfDay       = 0000-2400
ActionScript    = "/usr/sbin/rsct/bin/wallevent"
ReturnCode      = 0
CheckReturnCode = "n"
EventType       = "b"
StandardOut     = "n"
EnvironmentVars = ""
UndefRes       = "n"

```

13.3 Customizing CSM Blue Gene monitoring capabilities

In 13.2, “Monitoring the Blue Gene/P database with CSM” on page 180, we discussed CSM Blue Gene monitoring capabilities and mentioned the predefined conditions and responses that come with CSM. As powerful and useful as these are, CSM cannot deliver all the predefined conditions and responses that you might need, but you can customize CSM for your needs. However, before we discuss the commands that you can use to customize CSM Blue Gene monitoring capabilities, we need to describe in more detail the whole CSM Blue Gene/P database monitoring story.

To simplify the earlier monitoring discussion, we purposely neglected to mention a few things. If you examine the output of `1scondition` BGNODEErr (Example 13-1), you will notice that there is no mention of the TBGPNode table or our interest in row updates that set the status column to *E* or *M*. So where is this encoded? And how does the monitoring of the Blue Gene database really work?

Figure 13-1 shows the flow of CSM. At the root are the Blue Gene/P *database* and the *control system software* that writes to the database. Everything above the database comes with CSM or is created by CSM when you run various commands. At the top are a couple of CSM monitoring *constructs* that we talked about already: a *response* and a *condition*. Below these is a CSM monitoring construct called a *sensor*, which we will talk about soon. And below the sensor are two DB2 constructs: a Stored Procedure and a Trigger. The sensor, condition, and response used can be defined by you, predefined or a combination of both. We will talk about defining your own shortly.

The Trigger and Stored Procedure are created automatically for you by CSM when you start monitoring with the `startcondresp` command we already mentioned. The large up-pointing arrow on the right simply indicates the overall flow; in layman's terms, the Trigger watches the database for a specified event to happen. If and when that event happens, the Trigger gathers pertinent data and passes it to the Stored Procedure. The Stored Procedure is just a middleman that passes the data to the sensor. When the condition becomes aware of new data in the sensor, the condition evaluates its EventExpression, which is based on sensor data. If the EventExpression evaluates true, an event is generated and passed to the response. The response puts up a wall message, sends an e-mail, runs a script, or whatever it is configured to do.

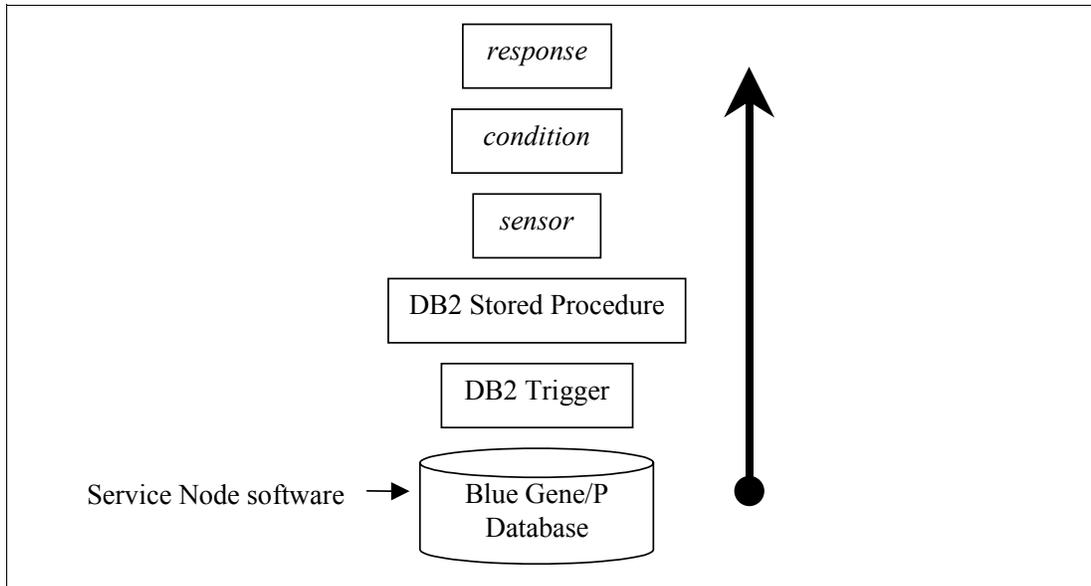


Figure 13-1 CSM flow

Before introducing the CSM commands used to create custom monitoring constructs, let's take a closer look at the constructs called sensors. As Figure 13-1 shows, a condition is paired with a sensor. Look again at the output of `lscollection BGNodeErr` in Example 13-1 on page 181. Two attributes specify which sensor the condition `BGNodeErr` is paired with:

```
ResourceClass    = "IBM.Sensor"
SelectionString  = "Name==\"BGNodeErr\""
```

The condition `BGNodeErr` is paired with a sensor of the same name. Details about the sensor can be obtained by running `lssensor sensor_name` as shown in Example 13-3.

Example 13-3 The lssensor command

```
# lssensor BGNodeErr
Name = BGNodeErr
ActivePeerDomain =
Command = /opt/csm/csmbin/bgmanage_trigger -t TBGPNODE -o u
-w STATUS -x "n.STATUS = 'E' OR n.STATUS = 'M'"
-p PRODUCTID,TBGPPRODUCTTYPE.DESCRPTION,MIDPLANEPOS,
  NODECARDPOS,POSITION,STATUS
-g MIDPLANEPOS,NODECARDPOS,POSITION
-T TBGPNODE -O u -W STATUS
-X "n.STATUS='A'"
-G MIDPLANEPOS,NODECARDPOS,POSITION
-P PRODUCTID,TBGPPRODUCTTYPE.DESCRPTION,MIDPLANEPOS,
  NODECARDPOS,POSITION,STATUS BGNodeErr
ConfigChanged = 0
ControlFlags = 5
Description = This sensor is updated when the node status is "E" (for error) or
"M" (for missing) in the Blue Gene system. The sensor is also updated when the
node status changes back to "A" (for available). Use "SD.Uint32>0 && SD.Int32==1"
as the event expression and "SD.Uint32>0 && SD.Int32==-1" as the rearm expression
in all the corresponding conditions.
ErrorExitValue = 1
ErrorMessage =
```

```

ExitValue = 0
Float32 = 0
Float64 = 0
Int32 = 0
Int64 = 0
NodeNameList = {bgpdd1sys1}
RefreshInterval = 0
SavedData =
SD = [,0,0,0,0,0,0]
String =
TimeCommandRun = Thu Sep 27 09:12:05 2007
Uint32 = 0
Uint64 = 0
UserName = bgpsysdb

```

Here, you see that sensor `BGNodeErr`'s `Command` attribute identifies `TBGPNode` as the Blue Gene table of interest, and `n.STATUS = 'E'` OR `n.STATUS = 'M'` as the column values to watch for. When you start monitoring, the sensor's `Command`, `bgmanage_trigger`, is called with the arguments shown. It is the job of `bgmanage_trigger` to create the DB2 Trigger and Stored Procedure required.

The purpose of this exposé is to point out that there are three main CSM monitoring constructs involved in monitoring the Blue Gene/P database - a sensor, a condition, and a response. There are many predefined ones you can use; but if these don't meet your needs entirely, you can define your own.

13.3.1 Defining your own CSM monitoring constructs

In general, custom sensors are defined with the CSM `mksensor` command. However, for Blue Gene sensors, you need to use the CSM `bgmksensor` command instead because it understands the Blue Gene database and provides all the flags and options needed for creating a Blue Gene database sensor. Custom conditions and responses are defined using the standard CSM `mkcondition` and `mkresponse` commands.

For example, suppose you want to monitor the `TBGPFanEnvironment` table for high fan temperatures. CSM provides no predefined sensor for this. However, it is easy to define your own. On the Service Node, you could run:

```
bgmksensor -t TBGPFanEnvironment -o i -x "n.temperature>35"
-p \${DEFAULT},temperature BGFanTempHi
```

Translation: Create a Blue Gene sensor named `BGFanTempHi`. Whenever a row gets inserted in the `TBGPFanEnvironment` table with a temperature above 35 degrees Celsius, the sensor should cache the fan location and temperature and notify all conditions that care. (The values to cache are specified with the `-p` flag, and are the values passed to the response in the generated event notification.)

Your next step would be to create a condition to monitor this new sensor. For example, you would run a command like this on your management server:

```
mkcondition -d 'Generate an event when the temperature of a Blue
Gene fan module rises above 35 degrees Celsius.' -r IBM.Sensor -s
'Name="BGFanTempHi"' -m 1 -e "SD.Uint32>0 && SD.Int32==1" BGFanTempHi
```

Translation: Create a condition named BGFanTempHi to monitor the sensor with the same name. (The `-m` and `-e` flags must simply be set as shown for all conditions created to monitor Blue Gene sensors).

To start monitoring, run the following commands on your management server:

```
mkcondresp BGFanTempHi MsgEventsToRootAnyTime "E-mail root off-shift"
LogCSMEventsAnyTime
startcondresp BGFanTempHi
```

When you have completed these steps, CSM will have created the necessary DB2 Triggers and Stored Procedure for you and turned monitoring on for condition BGFanTempHi. To verify that all is well, there are two things you can check:

- ▶ As root, run: `1saudrec -1`

Near or at the bottom of the output, you should see information like this:

```
Time          = 07/27/07 16:39:23 274743
Subsystem     = ERRM
Category      = Info
Description   = Monitoring of condition BGFanTempHi is started successfully.
```

- ▶ As the user `bgpsysdb` with a connection to `bgdb0`, run the following query:

```
db2 "select trigname from syscat.triggers where trigname = 'BGFANTEMPHICSME'"
```

This should return:

```
TRIGNAME
-----
BGFANTEMPHICSME
```

1 record(s) selected.

Then run the following query:

```
db2 "select procname,text from syscat.procedures where procname like
'COMMON_BGP%'"
```

Which should return:

```
PROCNAME
-----
COMMON_BGP
COMMON_BGP_EXT
```

2 record(s) selected.

With monitoring on, each time a row is added to the `TBGPFanEnvironment` table with a temperature above 35 degrees Celsius, an event will be generated. Due to the `mkcondresp` command above, the event will kick off three responses on the management server: a message to root announcing the event, an e-mail to root with event details if the event occurs off-shift, and a logging of the event in the `/var/log/csm/systemEvents` file. The message to root will look like Example 13-4.

Example 13-4 Message sent to root

```
Message from root@c96m5sn02 on <no tty> at 15:01 ...
Critical Event occurred:
Condition: BGFanTempHi
Node: c96m5sn02.ppd.pok.ibm.com
Resource: BGFanTempHi
```

```

Resource Class: Sensor
Resource Attribute: SD
Attribute Type: CT_SD_PTR
Attribute Value: [TBGPFANMODULE.PRODUCTID=39J5051|:|
TBGPFANMODULE.TBGPPRODUCTTYPE.DESRIPTION=Fan Module|:|
TBGPFANMODULE.MIDPLANEPOS=R00-M0|:|
LOCATION=A0|:|TEMPERATURE=36,0,1,0,0,0,0]
Time: Friday 07/21/07 15:00:59

```

The e-mail and log entry looks like Example 13-5.

Example 13-5 Message sent in e-mail

Friday 07/21/07 15:00:59

```

Condition Name: BGFanTempHi
Severity: Critical
Event Type: Event
Expression: SD.Uint32>0 && SD.Int32==1

```

```

Resource Name: BGFanTempHi
Resource Class: IBM.Sensor
Data Type: CT_SD_PTR
Data Value: ["TBGPFANMODULE.PRODUCTID=39J5051|:|
TBGPFANMODULE.TBGPPRODUCTTYPE.DESRIPTION=Fan Module|:|
TBGPFANMODULE.MIDPLANEPOS=R00-M0|:|
LOCATION=A0|:|TEMPERATURE=36",0,1,0,0,0,0]
Node Name: c96m5sn02.ppd.pok.ibm.com
Node NameList: {c96m5sn02.ppd.pok.ibm.com}
Resource Type: 0

```

13.3.2 Miscellaneous related information

If you are interested in examining the DB2 Trigger and Stored Procedure that CSM creates for you, run **bgmksensor** with the **-v** flag (**bgmksensor** formulates the SQL statements used to create the Trigger and Stored Procedure before they are actually needed so that it can test them), or query the Blue Gene database directly (after you have started monitoring with the **startcondresp** command).

The Trigger name is derived from the sensor name by appending *CSMe*. For example, for a sensor named BGFanTempHi, there will be a Trigger named BGFanTempHiCSMe when that sensor is actively used in monitoring.

The Stored Procedure story is actually more complicated than we have described here. We create two Stored Procedures. One is named COMMON_BGP, and the other COMMON_BGP_ext. The Trigger calls COMMON_BGP which calls COMMON_BGP_ext. COMMON_BGP exists to catch any SQL exceptions that occur. COMMON_BGP_ext calls a utility named **refresh_sensor** in a shared library named bgrefresh_sensor.so. **refresh_sensor** writes the data into the sensor.

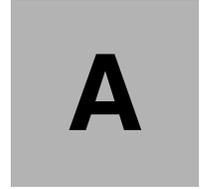
We create yet another DB2 construct that we have not mentioned, because it plays a minor role—a DB2 Sequence. Its named is derived from the sensor name by appending *_CSM*. For example, for a sensor named BGFanTempHi, there will be a Sequence named BGFanTempHi_CSM. The Trigger uses the Sequence to obtain a new number for each event it forwards to COMMON_BGP.

If you have started monitoring, you can connect to the database (with the administrator profile) and run queries similar to the following to examine the details of the DB2 triggers, stored procedures and sequences that CSM creates for you:

```
db2 connect to bgdb0 user bgpsysdb using xxxxxx (xxxxx= admin password)
db2 "select text from syscat.triggers where trigname = 'BGFanTempHiCSMe'"
db2 "select procname, text from syscat.procedures where procname like
'COMMON_BGP%'"
db2 "select * from syscat.sequences where seqname = ' BGFanTempHi_CSM'"
```

CSM brings powerful and customizable monitoring and automated response capabilities to the Blue Gene environment. By exploiting them you can minimize or eliminate much of the manual problem determination work often facing the Blue Gene system administrator. Furthermore, because these capabilities are extensions to existing CSM capabilities, you can easily monitor and set up automated responses for non-Blue Gene database problems as well.

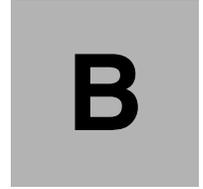
For example, get paged when the /var file system on your Service node fills up or send an urgent e-mail to the appropriate person when the number of users on a Front End node crosses some threshold. Run a script when the network adapter on a File Server is being overwhelmed. And there are so many other examples. Moving beyond monitoring, CSM offers a wealth of other capabilities that could help you manage your Blue Gene systems, such as distributed command execution, configuration file management, and software maintenance.



Statement of completion

IBM considers installation to be complete when the following activities have taken place:

- ▶ The Blue Gene/P rack or racks have been physically placed in position.
- ▶ The cabling is complete, including power, ethernet, and torus cables.
- ▶ The power to the Blue Gene/P racks can be turned on.
- ▶ All hardware is displayed in the Navigator and available.



Blue Gene/P hardware naming convention

This appendix provides an overview of how the Blue Gene/P hardware locations are assigned. This naming convention is used consistently throughout both the hardware and software.

Hardware naming conventions used in Blue Gene/P

Figure B-1 shows the conventions used when assigning locations to all hardware except the various cards in a Blue Gene/P system.

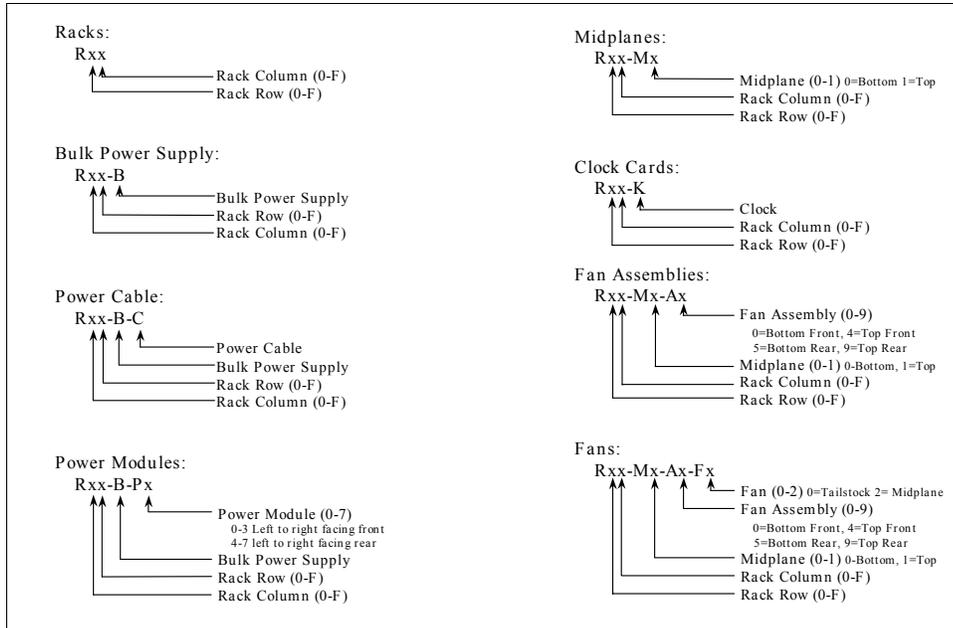


Figure B-1 Hardware naming conventions

As an example, if you had an error in fan R23-M1-A3-0 where would you go to look for it? In the upper left corner of Figure B-1 you see racks use the convention Rxx. Looking at our error message we can see that the rack involved is R23. From the chart we can see that is the fourth rack in row two (remember that all numbering starts with 0). The bottom midplane of any rack is 0, so we are dealing with the top midplane (R23-M1). In the chart you can see in the Fan Assemblies description that assemblies zero through four are on the front of the rack, bottom to top, respectively. So we are going to be checking for an attention light (Amber LED) on the Fan Assembly second from the top, because the front most fan is the one causing the error message to surface. Service, Link, and Node cards use a similar form of addressing.

Figure B-2 shows the conventions used for the various card locations.

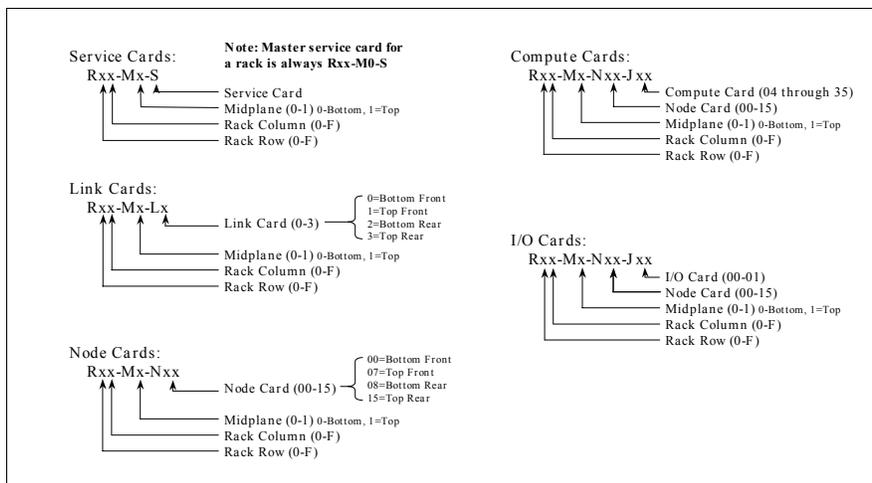


Figure B-2 Card naming conventions

Table B-1 contains examples of various hardware conventions. The figures that follow the table better illustrate the actual hardware.

Table B-1 Example of names

Card	Element	Name	Example
Compute	Card	J04 through J35	R23-M10-N02-J09
I/O	Card	J00 through J01	R57-M1-N04-J00
I/O & Compute	Module	U00	R23-M0-N13-J08-U00
Link	Module	U00 through U05 (00 left most, 05 right most)	R32-M0-L2_U03
Link	Port	TA through TF	R01-M0-L1-U02-TC
Link data cable	Connector	J00 through J15 (as labeled on link card)	R21-M1-L2-J13
Node Ethernet	Connector	EN0, EN1	R16-M1-N14-EN1
Service	Connector	Control FPGA, Control Network *, Clock R, Clock B	R05-M0-S-Control FPGA
Clock	Connector	Input, Output 0 through Output 9	R13-K- Output 3

Figure B-3 shows the layout of a 64 rack system.

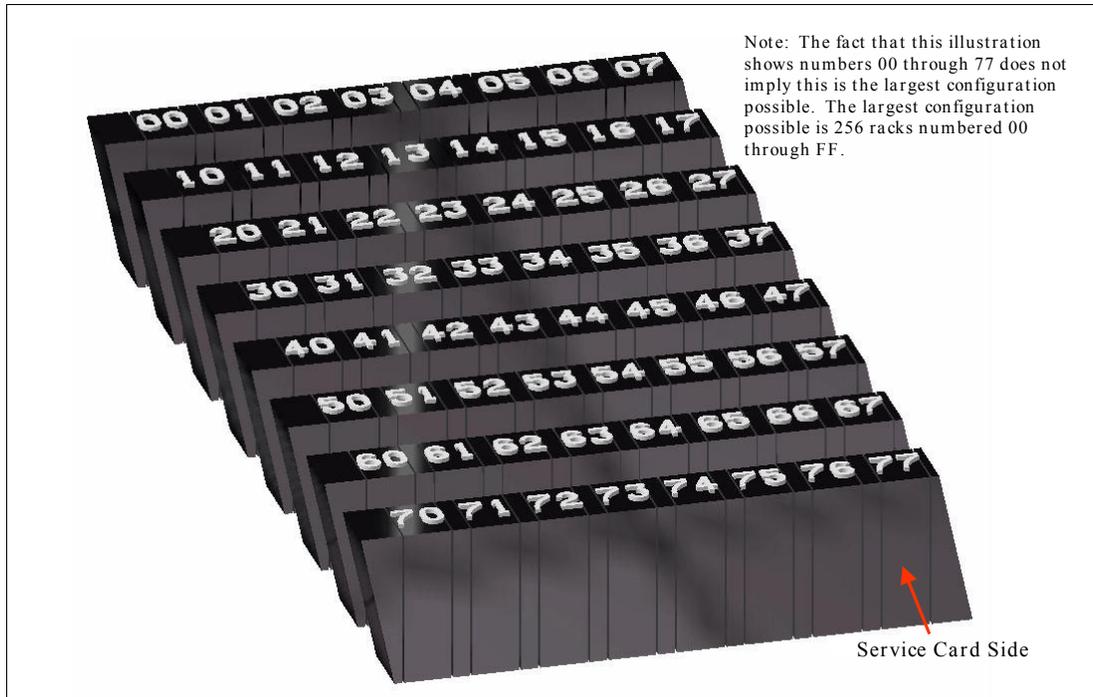


Figure B-3 Rack numbering

Figure B-4 identifies each of the cards in a single midplane.

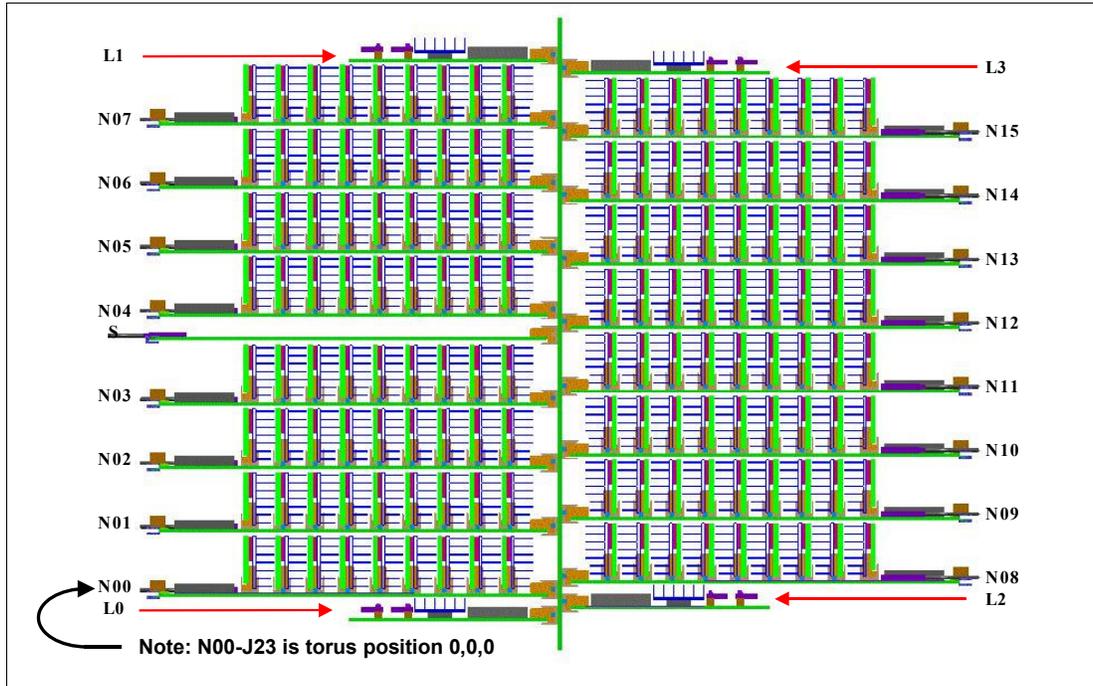


Figure B-4 Positions of Node, Link and Service cards

Figure B-5 is a diagram of a node card. On the front of the card are the ethernet ports EN0 and EN1. The first two nodes behind the ethernet ports are the I/O nodes. In this diagram the node card is fully populated with I/O nodes, meaning that it has two I/O nodes. Behind the I/O nodes are the compute nodes.

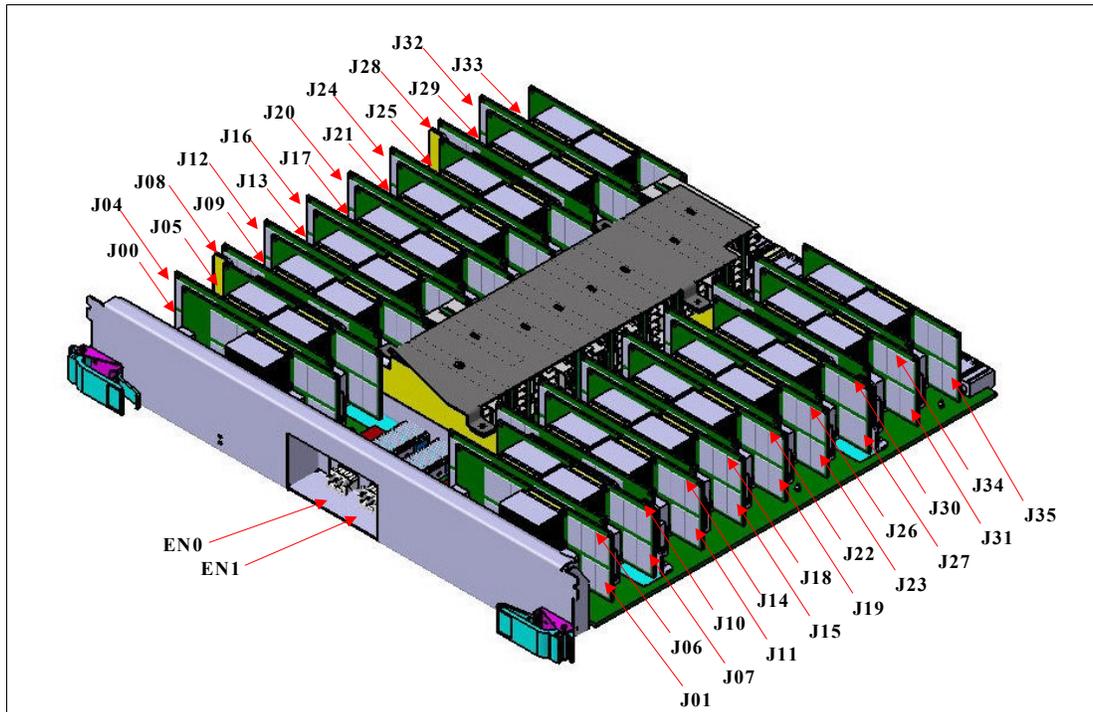


Figure B-5 Node card diagram

Figure B-6 is an illustration of a Service card.

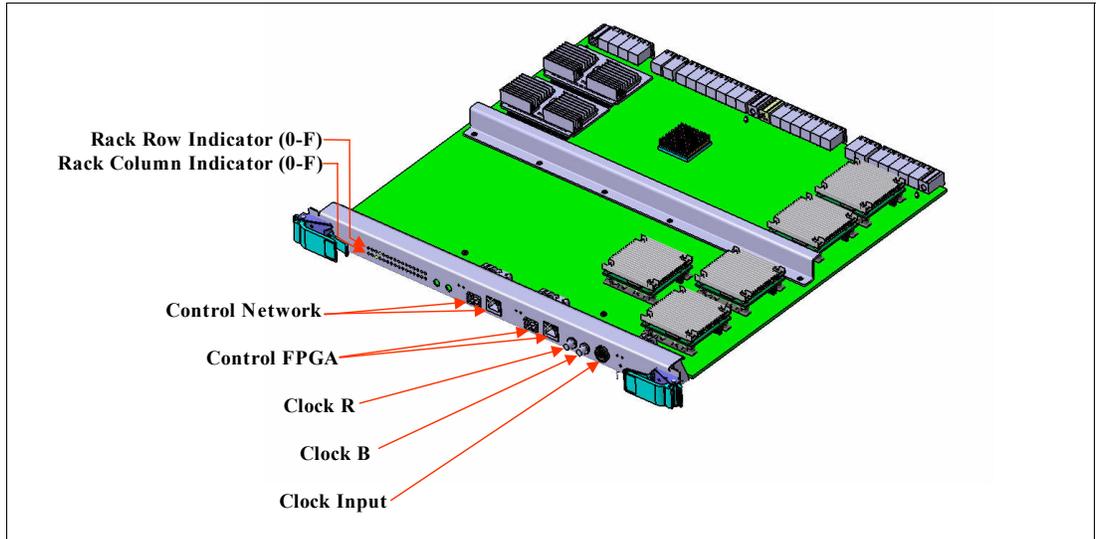


Figure B-6 Service Card

Figure B-7 shows the Link card. The locations identified as J00 through J15 are the link card connectors. The Link cables are routed from one link card to another to form the torus network between midplanes.

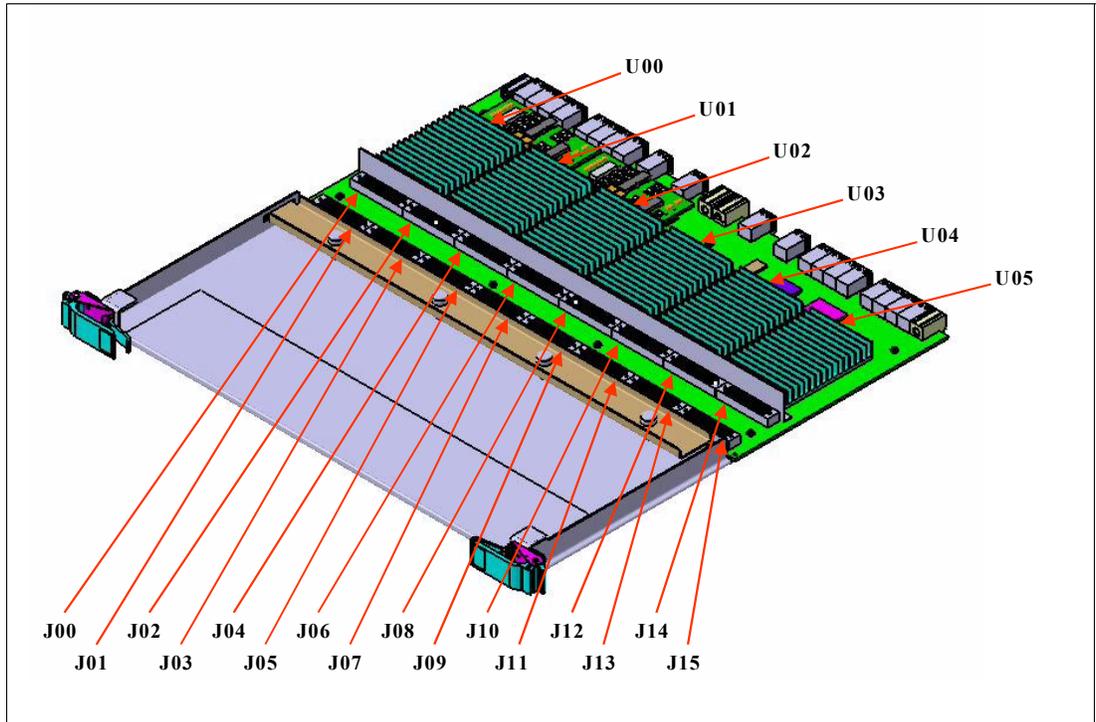


Figure B-7 Link card

Figure B-8 shows the clock card. If the clock is a secondary or tertiary clock there will be a cable coming to the input connector on the far right. Next to the input (just to the left) is the Master/Slave toggle switch. All clock cards are built with the capability of filling either role. If the clock is a secondary or tertiary clock, this must be set to *slave*. Output zero through nine can be used to send signals to midplanes throughout the system.

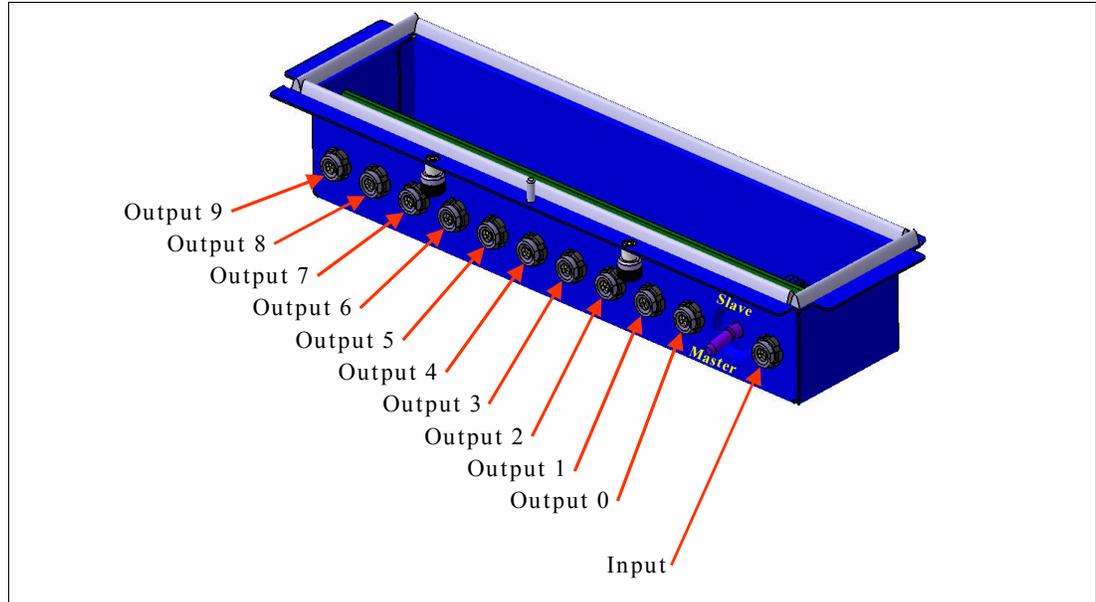


Figure B-8 Clock card

Glossary

32b executable Executable binaries (user applications) with 32b (4B) virtual memory addressing. Note that this is independent of the number of bytes (4 our 8) utilized for floating-point number representation and arithmetic.

32b floating-point arithmetic Executable binaries (user applications) with 32b (4B) floating-point number representation and arithmetic. Note that this is independent of the number of bytes (4 our 8) utilized for memory reference addressing.

32b virtual memory addressing All virtual memory addresses in a user application are 32b (4B) integers. Note that this is independent of the type of floating-point number representation and arithmetic.

64b executable Executable binaries (user applications) with 64b (8B) virtual memory addressing. Note that this is independent of the number of bytes (4 our 8) utilized for floating-point number representation and arithmetic. Note that all user applications should be compiled, loaded with Subcontractor supplied libraries and executed with 64b virtual memory addressing by default..

64b floating-point arithmetic Executable binaries (user applications) with 64b (8B) floating-point number representation and arithmetic. Note that this is independent of the number of bytes (4 our 8) utilized for memory reference addressing.

64b virtual memory addressing All virtual memory addresses in a user application are 64b (8B) integers. Note that this is independent of the type of floating-point number representation and arithmetic. Note that all user applications should be compiled, loaded with Subcontractor supplied libraries and executed with 64b virtual memory addressing by default.

AICF The Argonne Leadership Computing Facility operated for the U.S. Department of Energy's Office of Science by UChicago Argonne LLC.

API Application Programming Interface. Defines the syntax and semantics for invoking services from within an executing application. All APIs shall be available to both Fortran and C programs, although implementation issues (such as whether the Fortran routines are simply wrappers for calling C routines) are up to the supplier.

ASIC Application Specific Integrated Circuit.

ATP Acceptance Test Plan

b bit. A single, indivisible binary unit of electronic information.

B byte. A collection of eight (8) bits.

BGP Blue Gene/P. The name given to the collection of compute nodes, I/O nodes, front-end nodes, file systems, and interconnecting networks that is the subject of this statement of work.

BGPEA BGP Early Access platform

BPC BGP Compute ASIC. This high-function BGP ASIC is the basis of the compute nodes and I/O nodes.

BPL BGP Link ASIC. This high-function BGP ASIC is responsible for redriving communications signals between midplanes and is used to repartition BGP

CE On-site hardware customer engineer performing hardware maintenance.

cluster A set of nodes connected through a scalable network technology.

CMCS Cluster Monitoring and Control System.

CMN Control and Management Network. Provides a command and control path to BGP for functions such as, but not limited to, health status monitoring, repartitioning, and booting.

CN Compute Node. The element of BGP that supplies the primary computational resource for execution of a user application.

compute card One of the FRUs (field replaceable unit) of BGP. A compute card contains two complete compute nodes, and is plugged into a node card.

compute node (CN) The element of BGP that supplies the primary computational resource for execution of a user application.

Core The BGP Core is Subcontractor delivered hardware and software. The BGP Core consists of the BGP Compute Main Section, front-end node (FEN), service node (SN), and a control and management Ethernet.

CPU Central Processing Unit or processor. A VLSI chip constituting the computational core (integer, floating point, and branch units), registers and memory interface (virtual memory translation, TLB and bus controller).

current standard Applied when an API is not *frozen* on a particular version of a standard, but shall be upgraded automatically by the Subcontractor as new specifications are released. For example, *MPI version 2.0* refers to the standard in effect at the time of writing this document, while *current version of MPI* refers to further versions that take effect during the lifetime of this contract.

CWFS Cluster Wide File System. The file system that is visible from every node in the system with scalable performance.

DDR Double Data Rate. A technique for doubling the switching rate of a circuit by triggering on both the rising edge and falling edge of a clock signal.

EDRAM Enhanced dynamic random access memory is dynamic random access memory that includes a small amount of static RAM (SRAM) inside a larger amount of DRAM. Performance is enhanced by organizing so that many memory accesses will be to the faster SRAM.

FEN Front-End Node. The front-end node is responsible, in part, for interactive access to Blue Gene/P.

FGES Federated Gigabit-Ethernet Switch. The FGES connects the I/O nodes of Blue Gene/P to external resources, such as the FEN and the CWFS.

FLOP or OP Floating Point Operation

FLOPS or OPS Plural of FLOP

FLOPs or OPs Floating Point Operations per second.

front-end node The front-end node (FEN) is responsible, in part, for interactive access to Blue Gene/P

FRU Field Replaceable Unit.

fully supported Refers to product-quality implementation, documented and maintained by the HPC machine supplier or an affiliated software supplier.

GB gigabyte. A billion base 10 bytes. This is typically used in every context except for Random Access Memory size and is 109 (or 1,000,000,000) bytes.

GiB gibibyte. A billion base 2 bytes. This is typically used in terms of Random Access Memory and is 230 (or 1,073,741,824) bytes.

GFlops or GOPs gigaflops. A gigaflops is a billion (109 = 1,000,000,000) 64-bit floating point operations per second.

host complex The host complex includes the front-end node (FEN) and service node (SN)

HSN Hot Spare Node.

ICON The ICON is a high-function BGP ASIC that is responsible for Ethernet-to-JTAG conversion and other control functions.

IP The Internet Protocol (IP) is the method by which data is sent from one computer to another on the Internet.

job A cluster wide abstraction similar to a POSIX session, with certain characteristics and attributes. Commands shall be available to manipulate a job as a single entity (including kill, modify, query characteristics, and query state).

I/O I/O (input/output) describes any operation, program, or device that transfers data to or from a computer.

I/O card One of the FRUs (field replaceable unit) of BGP. An I/O card contains two complete I/O nodes, and is plugged into a node card.

I/O node The I/O nodes (ION) are responsible, in part, for providing I/O services to compute nodes.

IBM International Business Machines Corporation.

ION I/O node. Responsible, in part, for providing I/O services to compute nodes.

LINPACK LINPACK is a collection of Fortran subroutines that analyze and solve linear equations and linear least-squares problems.

Linux from Linus (Torvalds; program author's name) and UNIX®. Linux is a free Unix-like operating system originally created by Linus Torvalds with the assistance of developers around the world. Developed under the GNU General Public License, the source code for Linux is freely available to everyone.

MB megabyte. A million base 10 bytes. Typically used in every context except for Random Access Memory size and is 106 (or 1,000,000) bytes.

MiB mebibyte. A million base 2 bytes. Typically used in terms of Random Access Memory and is 220 (or 1,048,576) bytes. For a complete description of SI units for prefixing binary multiples see URL: <http://physics.nist.gov/cuu/Units/binary.html>

midplane An intermediate packaging component of BGP. Multiple node cards plug into a midplane to form the basic scalable unit of BGP.

MFlops or MOPs megaflops. A million (106 = 1,000,000) 64-bit floating point operations per second.

MPI Message Passing Interface.

MPICH2 MPICH is an implementation of the MPI standard available from Argonne National Laboratory.

MTBF Mean Time Between Failure. MTBF is a measurement of the expected reliability of the system or component. The MTBF figure can be developed as the result of intensive testing, based on actual product experience, or predicted by analyzing known factors. See URL: http://www.t-cubed.com/faq_mtbef.htm

node A node operates under a single instance of an operating-system image, and is an independent operating-system partition.

node card An intermediate packaging component of BGP. FRUs (field-replaceable units - compute cards and I/O cards) are plugged into a node card. Multiple node cards plug into a midplane to form the basic scalable unit of BGP.

OpenMP OpenMP is a portable, scalable model that gives shared-memory parallel programmers a simple and flexible interface for developing parallel applications.

peak rate The peak rate is the maximum number of 64-bit floating point instructions (add, subtract, multiply or divide) per second that could conceivably be retired by the system. For RISC CPUs the peak rate is typically calculated as the maximum number of floating point instructions retired per clock times the clock rate.

Ptrace A facility that allows a parent process to control the execution of a child process. Its primary use is for the implementation of breakpoint debugging.

published (as applied to APIs) This term shall refer to the situation where an API is not required to be consistent across platforms. An "published" API shall refer to the fact that the API shall be documented and supported, although it shall be Subcontractor, or even platform specific.

RAID Redundant Array of Independent (Inexpensive) Disks.

RAM Random Access Memory.

RAS Reliability, Availability and Serviceability.

SAN Storage Area Network, a high-speed subnetwork of storage devices.

scalable A system attribute that increases in performance or size as some function of the peak rating of the system. The scaling regime of interest is at least within the range of 1 teraflops to 60.0 (and possibly to 120.0) teraFLOPs peak rate.

service node The service node is responsible, in part, for management and control of Blue Gene/P.

SDRAM Synchronous, Dynamic Random Access Memory

single-point control (as applied to tool interfaces) The ability to control or acquire information about all processes/PEs using a single command or operation.

SMFS System Management File System. Provides a single, central location for administrative information about BGP.

SMP Symmetric Multi-Processor. A computing node in which multiple functional units operate under the control of a single operating-system image.

SN Service Node. Responsible, in part, for management and control of Blue Gene/P.

SPMD Single Program Multiple Data. SPMD is a programming model wherein multiple instances of a single program operates on multiple data.

sPPM The sPPM benchmark solves a 3D gas dynamics problem on a uniform Cartesian mesh, using a simplified version of the PPM (Piecewise Parabolic Method) code.

SRAM Static Random Access Memory.

standard (as applied to APIs) Where an API is required to be consistent across platforms, the reference standard is named as part of the capability. The implementation shall include all routines defined by that standard (even if some simply result in no-ops on a given platform).

TB terabyte. A terabyte is a trillion base 10 bytes. This is typically used in every context except for Random Access Memory size and is 10^{12} (or 1,000,000,000,000) bytes.

TCP/IP Transmission Control Protocol/Internet Protocol, the suite of communications protocols used to connect hosts on the Internet.

TiB tebibyte. A trillion bytes base 2 bytes. This is typically used in terms of Random Access Memory and is 240 (or 1,099,511,627,776) bytes. For a complete description of SI units for prefixing binary multiples see URL: <http://physics.nist.gov/cuu/Units/binary.html>

TFlops teraflops. A trillion ($10^{12} = 1,000,000,000,000$) 64-bit floating point operations per second.

UMT2000 The UMT benchmark is a 3D, deterministic, multigroup, photon transport code for unstructured meshes.

UPC Unified Parallel C, a programming language with parallel extensions to ANSI C. See, for example, <http://upc.gwu.edu/>

XXX-compatible (as applied to system software and tool definitions) This term shall require that a capability be compatible, at the interface level, with the referenced standard, although the lower-level implementation details will differ substantially. For example, "NFSv4-compatible" means that the distributed file system shall be capable of handling standard NFSv4 requests, but need not conform to NFSv4 implementation specifics.

Related publications

We consider the publications that we list in this section particularly suitable for a more detailed discussion of the topics that we cover in this book.

IBM Redbooks publications

For information about ordering these publications, see “How to get IBM Redbooks” on page 201. Note that some of the documents referenced here might be available in softcopy only.

Blue Gene/L applications:

- ▶ *Unfolding the IBM eServer Blue Gene Solution*, SG24-6686
- ▶ *Blue Gene/L: Performance Analysis Tools*, SG24-7278
- ▶ *IBM System Blue Gene Solution: Application Development*, SG24-7179

Hardware, Software, and System Administration:

- ▶ *IBM System Blue Gene Solution: System Administration*, SG24-7178
- ▶ *IBM System Blue Gene Solution: Configuring and Maintaining Your Environment*, SG24-7352
- ▶ *IBM System Blue Gene Solution: Hardware Installation and Serviceability*, SG24-6743
- ▶ *Blue Gene Safety Considerations*, REDP-4257
- ▶ *Blue Gene/L: Hardware Overview and Planning*, SG24-6796
- ▶ *GPFS Multicluster with the IBM System Blue Gene Solution and eHPS Clusters*, REDP-4168
- ▶ *IBM System Blue Gene Solution Problem Determination Guide*, SG24-7211
- ▶ *Evolution of the IBM System Blue Gene Solution*, REDP-4247

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Symbols

95
! 95
95

A

addbootoption 95
allocate 94–95
allocate_block 95
assignjobname 95
associatejob 95
async option 171
authentication 62

B

base partition (BP) 8
BG_BLOCKID 163
BG_CLOCKHZ 163
BG_FS 164
BG_GLINTS 164
BG_IP 172
BG_ISTORUS 164
BG_LOCATION 164
BG_MAC 164
BG_MTU 164
BG_NETMASK 164
BG_NODESINPSET 164
BG_NUMPSETS 164
BG_POWERMGMTDDRRATIO 176
BG_POWERMGMTDUR 176
BG_POWERMGMTPERIOD 176
BG_PSETNUM 164
BG_PSETSIZE 164
BG_SIMULATION 164
BG_SNIP 164
BG_UCI 164
BG_VERBOSE 164
BG_YSIZE 164
BG_ZSIZE 164
BGL_BLOCKID 172
BGL_BROADCAST 163
BGL_EXPORTDIR 164
BGL_GATEWAY 164
BGL_IP 164
BGL_LOCATION 172
BGL_VIRTUALNM 164
BGL_XSIZE 164
BGL_YSIZE 164
bgmsensor 184
BGPBLOCK 80
BGPMaster 14, 55, 63
 command 64
 mcServer 67

block

creating in mmcs 85
dynamic 80
genblock 85
genblocks 86
genBPblock 88
genfullblock 90
gensmallblock 87
mesh 80
pass-through 89
small 80
split data flow 89
static 80
block (partition) 80
block builder 13, 53, 81
blocks 8
Blue Gene Navigator 13
boot_block 94, 96
Bridge APIs 13
bulk power module (BPM) 2
BusyBox 162
 shell commands 162

C

CIOD 104, 165
 buffer 168
 buffer size 168
 environment variables 165
 service connection 167
CIOD_RDWR_BUFFER_SIZE 166
CIODB 10
 --ciod-persist 11
CioStream protocol 104
Cluster Systems Management (CSM) 179
collective network 5
compute card 3
compute node ratio 8
connect 96
Control and I/O Daemon (CIOD) 10
control network 5
copyblock 96
Coreprocessor tool 151
createjob 96
CSM
 bgmsensor 184
 lsresponse 181
 lssensor 183
 mkcondresp 180
 monitoring constructs 184
 startcondresp 181

D

data flow through CIOD 170
DataStream protocol 104

DB2 table

- BGPBLOCK 85
- BGPBULKPOWERENVIRONMENT 106
- BGPCLOCKCARDENVIRONMENT 106
- BGPEVENTLOG 106
- BGPFANENVIRONMENT 106
- BGPLINKCARDENVIRONMENT 106
- BGPLINKCARDPOWERENVIRONMENT 106
- BGPLINKCHIPENVIRONMENT 106
- BGPNODECARDENVIRONMENT 106
- BGPNODECARDPOWERENVIRONMENT 106
- BGPNODEENVIRONMENT 106
- BGPSERVICEACTION 143
- BGPSERVICECARDENVIRONMENT 106
- BGPSRVCCARDPOWERENVIRONMENT 106

dbPopulate 8

debugjob 96

delete 96

deselect_block 97

Diagnostics 15, 54, 126

Diagnostics test cases 126

Direct Memory Access (DMA) 5

disconnect 97

distribution directories 162

dual node mode 9

dynamic block 80

E

environment variable

- used by CIOD 165

Environmental Monitor 11, 104

Environmental Queries 48

envMonitor thread 105

F

fan module 3

file server 172

flow control 170

free 97

free_block 97

functional network 5

G

genblock 85, 97

genblocks 86, 97

genBPblock 88, 97

genfullblock 90, 97

gensmallblock 87, 97

getblockinfo 97

getjobinfo 97

getjobs 97

global barrier network 5

GPFS 173

GPFS (General Parallel File System)

- file server 166

H

hot_list_length 170

I

I/O Node

- log file 164

- performance 168

- site customization 170

- startup and shutdown scripts 162

I/O node 3

- mailbox 103

idoproxy 10

InstallServiceAction 8

J

JAAS tracing 62

job history 43

job_trace 98

K

K50ciod 170

kill_job 94, 98

killjob 98

klogd 167

L

link card 3

Link Summary 52

list (mmcs command) 98

list bgpjob 94

list_blocks 98

list_bps 98

list_jobs 98

list_users 99

locate (mmcs command) 99

locaterank 99

log viewer 45

lsresponse 181

lssensor 183

M

machine controller (mc) 10

mailbox 103

mcServer 10, 67, 93

mesh 80

message type 56

Midplane Management Control System (MMCS) 10, 70,

91

mkcondresp 180

mmcs 70

- clients 92

- commands 95

- console 93

- mcServer 93

- replyformat 102

- starting console session 93

MMCS Console 39

mounting additional file systems 168

mpiexec 13

mpirun 12

mpirun_be 12
Multiple Program Multiple Data (MPMD) 13

N

Navigator 18
 authentication 62
 BGPMaster 55, 67
 Block Builder 53, 81
 Blocks 39
 current jobs 38
 Diagnostics 54, 131
 Environmental Monitor 104
 Environmental Queries 48
 HealthCenter 35
 Job History 43
 Link Summary 52
 log viewer 45
 logging 61
 message type 56
 message types 57
 Midplane Activity 47
 MMCS console 39
 Pluggable Authentication Modules (PAM) 19
 plug-ins 60
 RAS Event Log 41
 resource links 57
 Service Actions 55, 142
 session data 34
 setup 18
 startup 21
 System Logs
 system logs 44
 system utilization 46
 user preferences 34
net.ipv4 169
netdev_max_backlog 170
network
 congestion 170
Network Time Protocol 167
NFS
 buffer sizes 171
 file server performance tips 169
 mount command 170
 mounts 170
 performance 169
NFS client 5

O

OpenMP 8
optmem_max 170

P

PAM module 62
partition 8
partition (block) 80
pass-through 89
personality.sh 163, 172
pipes command input 95

plenum 2
plug-in
 attention 59
 chart 59
 configuring 60
 creating 59
 installing 60
portmap 170
power management
 messages 177
 proactive 176
 reactive 176
processor set (pset) 164
pset (processor set) 164

Q

quit 94
quit (mmcs command) 99

R

RAS event log 41
RAS message type 57
read 166
reboot_nodes 99
receive socket buffer size 170
--reconnect 11
Redbooks Web site 201
 Contact us xi
redirect (mmcs command) 99, 103
regular expressions 11
Reliability, Availability, Serviceability (RAS) 8
replyformat (mmcs command) 99
resource link 57
rmem_default 170
rmem_max 170
rsize 171

S

S50ciod script 170
SACK support 170
Security Administration tool 150
select_block 94, 99
send socket buffer size 170
service (control) network 5
Service Action 14, 55, 140
set_username 93, 99
setblockinfo 99
setbootoptions 99
setdebuginfo 100
setjobargs 100
setusername 93, 100
shell commands implemented by BusyBox 162
shell variables 163
shutdown 171
 flow between rc scripts 163
sitefs script 171
skb-heads 170
sleep (mmcs command) 100

- small block 80
- socket
 - buffer size 170
- split data flow 89
- sql (mmcs command) 100
- startcondresp 181
- starthwpolling 100, 105
- startjob (mmcs command) 100
- startup and shutdown scripts
 - for I/O Node 162
- startup command 171
- static block 80
- status (mmcs command) 100
- stophwpolling 100
- submit_job 100
- submitjob 94, 101
- Symmetric Multi-Processing (SMP) mode 9
- syslogd 167
- system calls 166
- system utilization 46

T

- tcp option 170
- TCP timestamp support 169
- TCP window scaling support 170
- tcp_max_tw_buckets 169
- tcp_mem 170
- tcp_rmem 169
- tcp_timestamps 169
- tcp_window_scaling 170
- tcp_wmem 169
- threading 8
- torus network 5

U

- unmount 170
- username (mmcs command) 101

V

- VerifyCables 8
- version (mmcs command) 101
- virtual node mode (VNM) 9

W

- wait_boot 101
- wait_job 94, 101
- waitjob 94, 101
- wmem_default 170
- wmem_max 170
- write 166
- write_con 6
- write_con (mmcs command) 101
- wsize 171

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(->Hide:)->Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine:fm still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.

Draft Document for Review May 30, 2008 6:25 pm

7417spine.fm 207



IBM System Blue Gene Solution: Blue Gene/P System Administration

(1.5" spine)
1.5" <-> 1.998"
789 <-> 1051 pages



IBM System Blue Gene Solution: Blue Gene/P System Administration

(1.0" spine)
0.875" <-> 1.498"
460 <-> 788 pages



IBM System Blue Gene Solution: Blue Gene/P System Administration

(0.5" spine)
0.475" <-> 0.873"
250 <-> 459 pages



IBM System Blue Gene Solution: Blue Gene/P System Administration

(0.2" spine)
0.17" <-> 0.473"
90 <-> 249 pages

(0.1" spine)

0.1" <-> 0.169"
53 <-> 89 pages

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(->Hide:)->Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine:fm still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.

Draft Document for Review May 30, 2008 6:25 pm

7417spine.fm 208



IBM System Blue Gene Solution: Blue Gene/P System Administration

(2.5" spine)
2.5" <-> nnn.n"
1315 <-> nnnn pages



IBM System Blue Gene Solution: Blue Gene/P System Administration

(2.0" spine)
2.0" <-> 2.498"
1052 <-> 1314 pages



IBM System Blue Gene Solution: Blue Gene/P System Administration



Perform Blue Gene/P system administration

Learn how to configure and use Blue Gene Navigator

Run Diagnostics and perform Service Actions

This book is one in a series of IBM publications written specifically for the IBM System Blue Gene supercomputer, Blue Gene/P, which is the second generation of massively parallel supercomputer from IBM in the Blue Gene series. It provides an overview of the system administration environment for Blue Gene/P. It is intended to help administrators understand the tools that are available to maintain this system.

This book explains briefly the evolution of the system, from Blue Gene/L to the new Blue Gene/P. It details Blue Gene Navigator, which has grown to be a full featured Web-based system administration tool on Blue Gene/P.

The book also describes many of the day-to-day administrative functions, such as running Diagnostics, performing Service Actions, and monitoring hardware. There are also sections to cover BGPMaster and the processes that it monitors. The final chapters of the book cover the tools that are shipped with the system and details about configuring communications with the I/O nodes and Power Management™ features.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks