

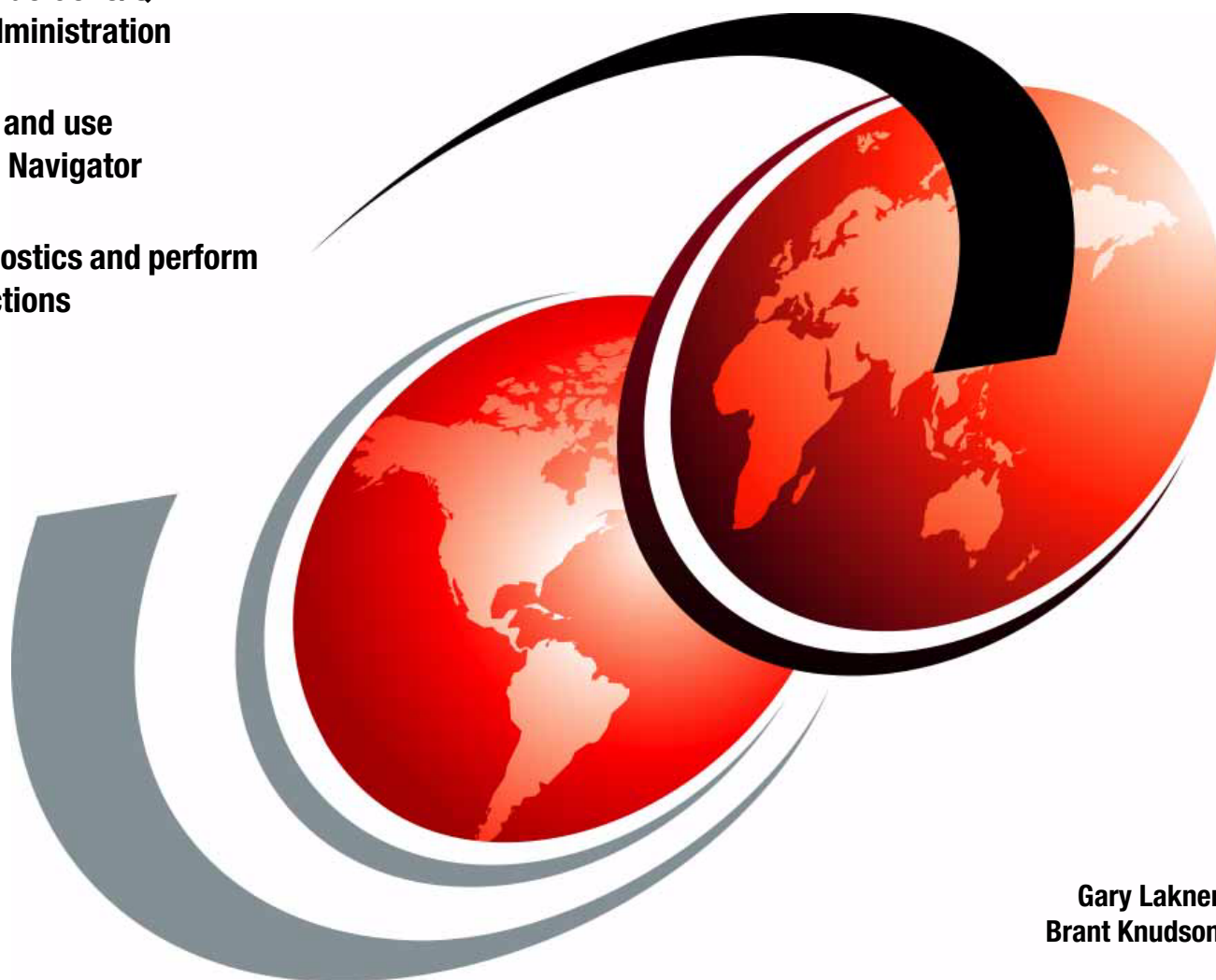
IBM System Blue Gene Solution

Blue Gene/Q System Administration

Perform Blue Gene/Q
system administration

Configure and use
Blue Gene Navigator

Run diagnostics and perform
service actions



Gary Lakner
Brant Knudson

Redbooks



International Technical Support Organization

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

September 2012

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

Second Edition (September 2012)

This edition applies to Version 1, Release 1, Modification 2 of IBM System Blue Gene/Q Solution (product number 5733-BGQ).

© Copyright International Business Machines Corporation 2012. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	xi
Trademarks	xii
Preface	xiii
The team who wrote this book	xiii
Now you can become a published author, too!	xiv
Comments welcome	xiv
Stay connected to IBM Redbooks	xiv
Summary of changes	xv
September 2012, Second Edition	xv
Chapter 1. IBM Blue Gene/Q system overview	1
1.1 Blue Gene/Q hardware overview	2
1.2 Blue Gene/Q software overview	3
1.2.1 Compute Node Kernel and services	3
1.2.2 I/O node kernel and services	3
1.2.3 Application development and debugging tools	4
1.2.4 System administration and management	5
Chapter 2. Navigator	7
2.1 System setup	8
2.2 Using the Navigator	9
2.2.1 The Navigator window	10
2.2.2 Security integration	11
2.2.3 Result tables	11
2.2.4 System summary	13
2.2.5 Alerts	13
2.2.6 Jobs and job history	14
2.2.7 Blocks and I/O blocks	15
2.2.8 RAS	17
2.2.9 Environmentals	18
2.2.10 Hardware	19
2.2.11 Hardware replacements	20
2.2.12 Diagnostics	21
2.2.13 Service actions	23
2.2.14 Performance monitoring	24
2.2.15 Knowledge Center	27
2.2.16 Creating blocks with the Block Builder	27
Chapter 3. Compute and I/O blocks	29
3.1 Compute blocks	30
3.1.1 Large blocks	30
3.1.2 Small blocks	34
3.2 Creating compute blocks	35
3.3 I/O blocks	37
3.4 Creating I/O blocks	37
3.5 I/O links and ratios	38
3.6 Deleting blocks	38

3.7	Block status transitions	38
Chapter 4. Configuring I/O nodes		
4.1	I/O node kernel	41
4.1.1	Building new Linux kernels	42
4.1.2	Installing the Linux kernel from the Red Hat download website	42
4.2	I/O node run time	43
4.2.1	Installing Red Hat Enterprise Linux	43
4.2.2	Installing additional packages	44
4.2.3	Creating Blue Gene/Q Linux distribution sandboxes	44
4.3	I/O adapters	45
4.3.1	Supported adapters	45
4.3.2	Firmware updates	45
4.4	I/O configuration	46
4.4.1	Network Interface MTU	46
4.4.2	Defining interfaces and IP addresses	46
4.4.3	Interface bonding configuration	48
4.4.4	Environment variables using /dev/bgpers	48
4.4.5	Site customization	50
4.4.6	Customizing ramdisk	50
4.4.7	The bgras command	51
4.5	Boot sequence	52
4.5.1	Persistent file systems	53
4.6	Maintenance	54
4.6.1	Installing RHEL updates	54
4.6.2	Installing a new RHEL dot release	54
4.6.3	Installing efixes	54
4.6.4	Installing GPFS updates	55
4.6.5	Node Health Monitor	56
4.6.6	Core files	57
4.7	Common I/O services	57
4.7.1	Configuration	58
4.7.2	Jobs directory	60
4.8	NFS configuration	61
4.9	Troubleshooting	63
4.9.1	NFS errors	63
Chapter 5. Control System console		
5.1	Overview	67
5.1.1	Accessing help	68
5.1.2	Command history	69
5.2	Commands	69
5.3	Special commands	72
5.4	Command parameters	72
5.5	Scripting	73
Chapter 6. Submitting jobs		
6.1	The runjob architecture	75
6.2	The runjob command	76
6.2.1	The runjob options	77
6.2.2	Scheduler	80
6.3	Sub-block jobs	80
6.4	Signal handling	82
6.4.1	Exit status	82

6.4.2	Normal job termination	82
6.4.3	Abnormal job termination	82
6.5	Standard input, output and error	83
6.6	Job authority	83
6.6.1	The grant_job_authority command	83
6.6.2	The revoke_job_authority command	84
6.6.3	The list_job_authority command	84
6.7	The kill_job command	85
6.8	Job status	86
6.8.1	The list_jobs command	88
6.8.2	The job_status command	89
6.9	Historical perspective of job submission on Blue Gene	90
Chapter 7. Reliability, availability, and serviceability		93
7.1	Elements of a RAS message	94
7.1.1	Message ID and component	94
7.1.2	Category	95
7.1.3	Severity	96
7.1.4	Message	96
7.1.5	Description	96
7.1.6	Service action	96
7.1.7	Diagnostics suites	97
7.1.8	Control Action	97
7.2	Tailoring RAS messages	98
7.3	Viewing Blue Gene/Q defined RAS events	99
7.4	Viewing sent RAS events	99
Chapter 8. Toolkit for Event Analysis and Logging		101
8.1	TEAL framework	102
8.1.1	Connector	103
8.1.2	Event analyzers	103
8.1.3	Plug-ins	104
8.2	Location reporting	104
8.3	Integration with BGmaster	104
8.4	Alerts and service actions	105
8.5	Installation	106
Chapter 9. Service actions		107
9.1	Service actions overview	108
9.2	Service action commands	108
9.2.1	InstallServiceAction	109
9.2.2	VerifyCables	109
9.3	Hardware maintenance	110
9.3.1	ServiceNodeDCA	111
9.3.2	ServiceNodeBoard	111
9.3.3	ServiceIoDrawer	111
9.3.4	ServiceMidplane	111
9.3.5	ServiceBulkPowerModule	112
9.3.6	ServiceRack	112
9.3.7	ServiceClockCard	113
9.4	Deciding which service action command to use	113
9.5	Preparing a service action	115
9.6	Ending a service action	116
9.7	Service action logs	117

9.8	Cycling power	117
Chapter 10. Diagnostics		
10.1	System diagnostics	120
10.1.1	Requirements	120
10.2	Diagnostic test cases	120
10.3	Using the Navigator diagnostics interface	123
10.4	Running diagnostics	123
10.5	Preventive maintenance	125
10.6	Running diagnostics through a scheduler	125
10.6.1	Running diagnostics from a LoadLeveler job	126
10.7	Diagnostics performance considerations	128
Chapter 11. BGmaster		
11.1	BGmaster	130
11.2	Running bgmaster_server	130
11.2.1	The master_start command	131
11.2.2	The master_status command	131
11.2.3	The binary_status command	132
11.2.4	The bgmaster_server_refresh_config command	133
11.2.5	The binary_wait command	133
11.2.6	The fail_over command	134
11.2.7	The alias_wait command	134
11.2.8	The list_agents command	135
11.2.9	The list_clients command	136
11.2.10	The master_stop command	136
11.2.11	The get_errors command	137
11.2.12	The get_history command	137
11.2.13	The monitor_master command	138
11.3	The bgagent daemon system process watcher	138
11.3.1	Starting bgagentd	139
11.3.2	Stopping bgagentd	139
11.4	Configuration	140
11.4.1	Sample configuration	141
11.4.2	BGmaster network configuration	143
11.5	Troubleshooting	144
11.5.1	Configuration issues	144
11.5.2	RAS messages	145
Chapter 12. Midplane Management Control System server		
12.1	MMCS components	148
12.1.1	MMCS server handles requests from clients	148
12.1.2	Blocks are defined in the database	148
12.1.3	MC server communicates with the Blue Gene/Q hardware	148
12.2	Starting and stopping MMCS server	149
12.3	Configuration	149
12.3.1	Configuration file options	149
12.3.2	Command-line options	150
12.4	Mailbox output	151
12.5	Environmental polling	151
12.5.1	How the environmental monitor works	152
12.5.2	Polling intervals	152
12.5.3	Data collection and RAS	153
12.5.4	Location-specific monitoring	156

12.5.5 RAS events	156
Chapter 13. Machine controller server	159
13.1 Machine controller server	160
13.2 Starting and stopping MC server.	160
13.3 Configuration.	160
13.3.1 Configuration file options	161
13.3.2 Command-line options	161
13.4 The mc_server_log_level command	161
Chapter 14. The runjob server and runjob mux	163
14.1 The runjob architecture	164
14.2 The runjob_server	165
14.2.1 Configuration file options	165
14.2.2 Command-line options	166
14.3 The runjob_mux	168
14.3.1 Configuration file options	168
14.3.2 Command-line options	170
14.4 Utilities.	171
14.4.1 Configuration options	171
14.4.2 The runjob_server_status command.	171
14.4.3 The runjob_server_log_level command	172
14.4.4 The runjob_mux_status command	174
14.4.5 The runjob_mux_log_level command	175
Chapter 15. The Blue Gene Web Services server	177
15.1 Configuring and running the BGWS server.	178
15.2 Configuration.	179
15.2.1 Configuration file options	179
15.2.2 Command-line options	180
15.3 Utilities.	180
15.3.1 Configuration.	180
15.3.2 The bgws_server_status command	181
15.3.3 The bgws_server_refresh_config command.	181
15.3.4 The bgws_server_log_level command	182
Chapter 16. Real-time server	185
16.1 Real-time server	186
16.2 Configuration.	186
16.2.1 Configuration file options	186
16.2.2 Command-line options	187
16.3 Security	187
16.4 Utilities.	187
16.4.1 Configuration.	188
16.4.2 The realtime_server_status command	188
16.4.3 The realtime_server_log_level command.	189
16.5 Requirements	189
16.5.1 Access to the database transaction logs	189
16.5.2 Database configuration.	190
Chapter 17. Distributed Control System	193
17.1 Overview	194
17.2 Distributed Control System software.	194
17.3 Hardware configurations	195

17.3.1	Multiple SubnetMc processes on a service node	196
17.3.2	Multiple subnet service nodes	197
17.3.3	Multiple subnet service nodes with failover	198
17.4	SubnetMc configuration	199
17.4.1	Troubleshooting configuration issues	201
Chapter 18. Security	203
18.1	Object based model	204
18.2	Network communication security	205
18.2.1	Certificates and keys	205
18.2.2	Handshake	206
18.2.3	Configuration	207
18.2.4	Certificate file permissions	208
18.2.5	Generating security keys and certificates	209
18.3	Database security	210
Chapter 19. Database	211
19.1	Database overview	212
19.2	Database configuration	212
19.3	Database maintenance	212
19.3.1	Purging old data from tables	213
19.3.2	Optimizing with indexes	214
19.3.3	Reorganizing tables	216
Chapter 20. Logging	219
20.1	Apache log4j and log4cxx	220
20.2	Log file format	220
20.3	Log merge utility	221
20.3.1	Configuration	221
20.3.2	Usage	222
20.3.3	Time stamp format	222
20.4	Configuration	223
20.5	Log rotation	223
20.5.1	Control System logs	224
20.5.2	I/O node logs	224
20.5.3	Subnet service node logs	224
20.5.4	Log maintenance	225
Chapter 21. The bg.properties file	227
21.1	The bg.properties file overview	228
21.2	Property file validation	230
Chapter 22. The Coreprocessor tool	231
22.1	Usage and dependencies	232
22.2	Starting the Coreprocessor tool	232
22.3	Debugging live compute node problems	234
22.4	Saving your information	237
22.5	Debugging live I/O node problems	238
22.6	Debugging core files	238
Appendix A. Hardware location naming conventions	241
Letter designation reference		241
Hardware location naming convention		242
Appendix B. Control System simulator	247

Setting up	248
Starting and stopping servers	249
Creating and booting I/O blocks and compute blocks	250
Running jobs	250
Limitations with runjob	250
Simulator cleanup	251
Related publications	253
IBM Redbooks	253
Other publications	253
Online resources	253
How to get Redbooks	254
Help from IBM	254
Abbreviations and acronyms	255
Index	257

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.

IBM DOES NOT WARRANT OR REPRESENT THAT THE CODE PROVIDED IS COMPLETE OR UP-TO-DATE. IBM DOES NOT WARRANT, REPRESENT OR IMPLY RELIABILITY, SERVICEABILITY OR FUNCTION OF THE CODE. IBM IS UNDER NO OBLIGATION TO UPDATE CONTENT NOR PROVIDE FURTHER SUPPORT.

ALL CODE IS PROVIDED "AS IS," WITH NO WARRANTIES OR GUARANTEES WHATSOEVER. IBM EXPRESSLY DISCLAIMS TO THE FULLEST EXTENT PERMITTED BY LAW ALL EXPRESS, IMPLIED, STATUTORY AND OTHER WARRANTIES, GUARANTEES, OR REPRESENTATIONS, INCLUDING,

WITHOUT LIMITATION, THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF PROPRIETARY AND INTELLECTUAL PROPERTY RIGHTS. YOU UNDERSTAND AND AGREE THAT YOU USE THESE MATERIALS, INFORMATION, PRODUCTS, SOFTWARE, PROGRAMS, AND SERVICES, AT YOUR OWN DISCRETION AND RISK AND THAT YOU WILL BE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY RESULT, INCLUDING LOSS OF DATA OR DAMAGE TO YOUR COMPUTER SYSTEM.

IN NO EVENT WILL IBM BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY OR CONSEQUENTIAL DAMAGES OF ANY TYPE WHATSOEVER RELATED TO OR ARISING FROM USE OF THE CODE FOUND HEREIN. WITHOUT LIMITATION, ANY LOST PROFITS, BUSINESS INTERRUPTION, LOST SAVINGS, LOSS OF PROGRAMS OR OTHER DATA, EVEN IF IBM IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS EXCLUSION AND WAIVER OF LIABILITY APPLIES TO ALL CAUSES OF ACTION, WHETHER BASED ON CONTRACT, WARRANTY, TORT OR ANY OTHER LEGAL THEORIES.


Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

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

Blue Gene/P®
Blue Gene/Q®
Blue Gene®
DB2®
eServer™

GPFS™
IBM®
LoadLeveler®
PowerPC®
Redbooks®

Redpapers™
Redbooks (logo) ®
System i®
Tivoli®

The following terms are trademarks of other companies:

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

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

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

Preface

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

This book details Blue Gene Navigator, which has grown to be a full featured web-based system administration tool on Blue Gene/Q. 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 that cover BGmaster and the Control System processes that it monitors.

This book is intended for Blue Gene/Q system administrators. It helps them use the tools that are available to maintain the Blue Gene/Q system.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization.

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

Brant Knudson is a Staff Software Engineer in the Advanced Systems SW Development group of IBM in Rochester, Minnesota, where he has been a programmer on the Control System team since 2003. Prior to working on Blue Gene, he worked on IBM Tivoli® Directory Server. Brant has been with IBM since 1998.

Special thanks to the following people for their contributions to this project:

Tom Budnik
Sam Miller

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

Jay Bryant
Cory Lappi
Pat McCarthy
Mark Megerian
Charlie Miller
Mike Mundy
Andrew Tauferner
William Stockdell
Yasunobu Suginaka
Kiswanto Thayib
Blue Gene Development, IBM Rochester

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

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 books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

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

ibm.com/redbooks

- ▶ Send your comments in an email 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

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

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

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 SG24-7869-01
for IBM System Blue Gene Solution: Blue Gene/Q System Administration
as created or updated on September 17, 2012.

September 2012, Second Edition

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

New information

- ▶ Added 4.4.1, “Network Interface MTU” on page 46.
- ▶ Added information about downloading and installing the Linux kernel from the Red Hat web site in 4.1.2, “Installing the Linux kernel from the Red Hat download website” on page 42.
- ▶ Added information about handling efixes for the bgq-distros RPM in 4.6.3, “Installing efixes” on page 54.
- ▶ Added new **fail_over** and **status** commands to the command table in Chapter 5, “Control System console” on page 67.
- ▶ Added information about the BGmaster **fail_over** command in 11.2.6, “The fail_over command” on page 134.
- ▶ Added new failover configuration information in Chapter 13, “Machine controller server” on page 159.
- ▶ Added information about reorganizing database tables in 19.3.3, “Reorganizing tables” on page 216.

Changed information

- ▶ Updated information for Mellanox PCIe adapter firmware in section 4.3.2, “Firmware updates” on page 45 to describe how updates are provided directly from IBM.
- ▶ Updated multiple Node Health Monitor threshold defaults in 4.6.4, “Installing GPFS updates” on page 55.
- ▶ Corrected and updated the **runjob** options table in Chapter 6, “Submitting jobs” on page 75.
- ▶ Updated various sections in Chapter 11, “BGmaster” on page 129 to reflect changes made in V1R1M2.
- ▶ Updated various sections in Chapter 12, “Midplane Management Control System server” on page 147 to reflect changes made in V1R1M2.
- ▶ Updated various sections in Chapter 17, “Distributed Control System” on page 193 to reflect changes made in V1R1M2 for failover.



IBM Blue Gene/Q system overview

The IBM Blue Gene/Q system, shown in Figure 1-1, is the third-generation computer architecture in the IBM Blue Gene family of supercomputers.

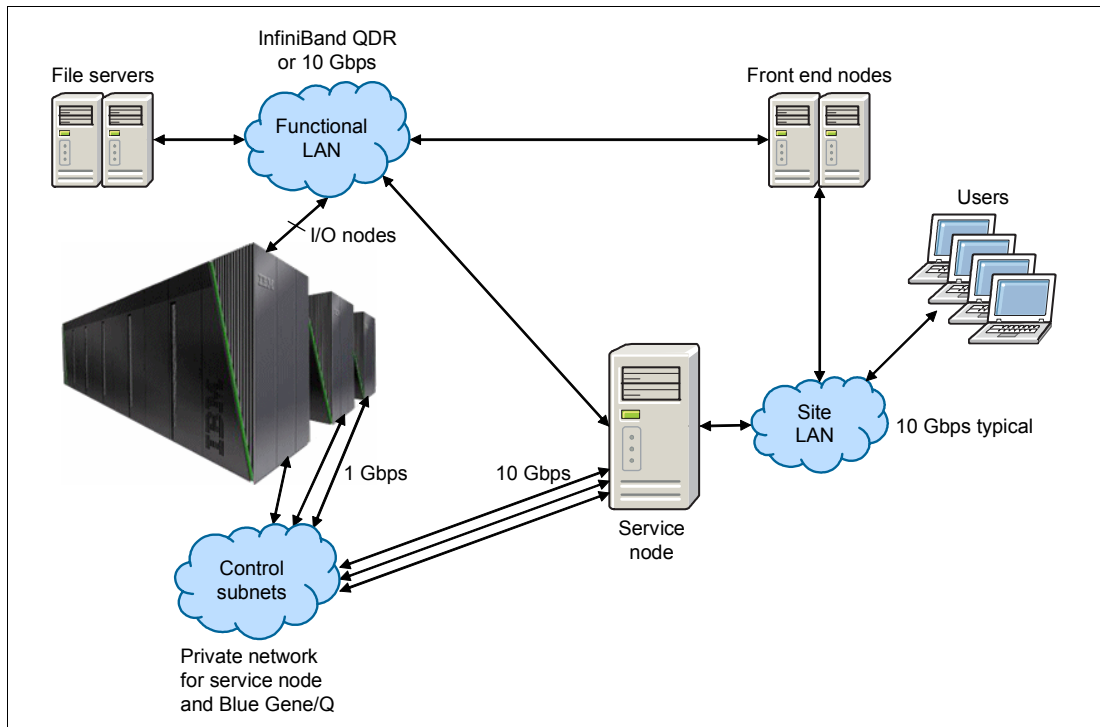


Figure 1-1 Blue Gene/Q system architecture

The Blue Gene/Q system comprises multiple components including one or more compute racks and optionally I/O racks. The system contains densely packaged compute nodes, I/O drawers, and service cards. Additional hardware is associated with the storage subsystem, the primary service node (SN), the front end nodes (FENs), and the communications

subsystem. The I/O drawers containing I/O nodes connect to the functional local area network (LAN) to communicate with file servers, FENs, and the SN. The service cards connect to the control subnets and are used by the SN to control the Blue Gene/Q hardware.

A service node provides a single point of control and administration for the Blue Gene/Q system. It is possible to operate a Blue Gene/Q system with a single service node. However, the system environment can also be configured to include a backup service node for failover capability and distributed subnet service nodes (SSN) for high scalability.

A front end node, also known as a login node, comprises the system resources that application developers login to for access to the Blue Gene/Q system. Application developers edit and compile applications, create job control files, launch jobs on the Blue Gene/Q system, post-process output, and perform other interactive activities. Application development is outside the scope of this book and is covered in the *IBM System Blue Gene Solution: Blue Gene/Q Application Development*, SG24-7948 Redbooks publication.

1.1 Blue Gene/Q hardware overview

Figure 1-2 shows the primary hardware components of Blue Gene/Q.

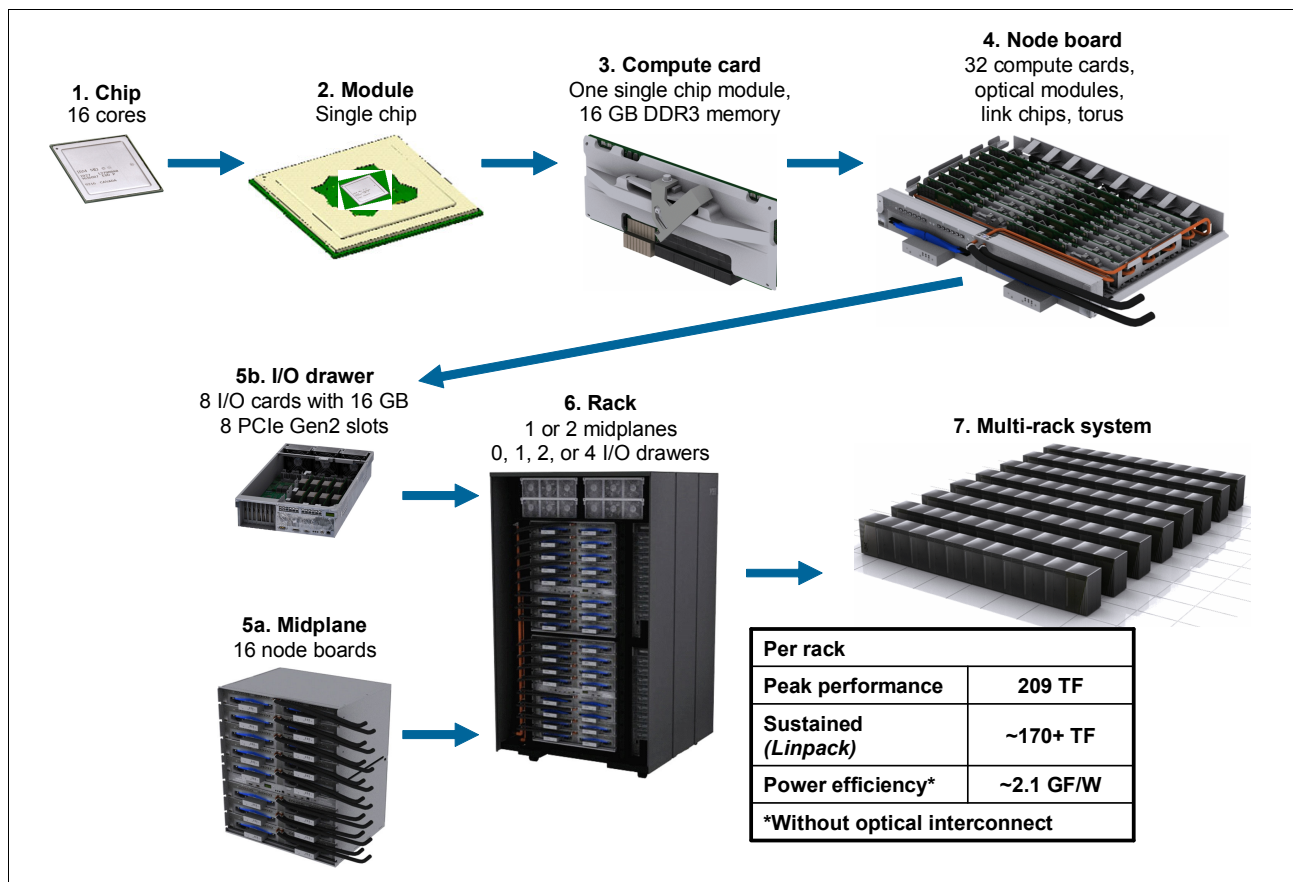


Figure 1-2 Blue Gene/Q hardware overview

Compute cards contain 16 IBM Blue Gene/Q PowerPC® A2 core processors and 16 GB of memory. Thirty-two such cards plug into a node board, and 16 node boards are contained in a midplane. A Blue Gene/Q compute rack has either one (half rack configuration) or two fully populated midplanes, and the system can be scaled to 512 compute racks.

Compute rack components are cooled either by water or air. Water is used for the processing nodes. Air is used for the power supplies and the I/O drawers mounted in the Blue Gene/Q rack.

I/O drawers are either in separate racks or in I/O enclosures on top of the compute racks, which are sometimes referred to as “top hats.” Eight I/O nodes are housed in each I/O drawer. In the compute rack, up to four I/O drawers, two per midplane, can be configured using the I/O enclosure (top hat). The placement of I/O drawers in the I/O rack configuration is advisable in a large system installation where the number of I/O nodes cannot be accommodated in the compute racks.

For an introduction to the Blue Gene/Q hardware components, see the *Blue Gene/Q Hardware Overview and Installation Planning Guide*, SG24-7822 Redbooks publication.

1.2 Blue Gene/Q software overview

The software for the Blue Gene/Q system includes the following integrated software subsystems:

- ▶ Compute Node Kernel (CNK) and services
- ▶ I/O node kernel and services
- ▶ Application development and debugging tools
- ▶ System administration and management
- ▶ Block and job management

The software subsystems are required in three hardware subsystems:

- ▶ Compute node
- ▶ I/O node
- ▶ Host complex (including front end node and service node)

1.2.1 Compute Node Kernel and services

The Compute Node Kernel (CNK) software is an operating system that is similar to Linux and provides an environment for running user processes on compute nodes. The CNK includes the following services:

- ▶ Process creation and management
- ▶ Memory management
- ▶ Process debugging
- ▶ Reliability, availability, and serviceability (RAS) management
- ▶ File I/O
- ▶ Network

A comprehensive description of the CNK is provided in the *IBM System Blue Gene Solution: Blue Gene/Q Application Development*, SG24-7948 Redbooks publication.

1.2.2 I/O node kernel and services

The I/O node kernel is a patched Red Hat Enterprise Linux 6 kernel running on I/O nodes. The patches provide support for the Blue Gene/Q platform and contain modifications to improve performance.

The I/O node software is responsible for providing I/O services to compute nodes. For example, applications running on compute nodes can access file servers and communicate

with processes in other machines. The I/O nodes also play an important role in starting and stopping jobs and in coordinating activities with debug and monitoring tools.

Blue Gene/Q is a diskless system so file servers must be present. A high-performance parallel file system is expected. Blue Gene/Q is flexible, accepting different file systems supported by Linux. Typical parallel file systems are IBM General Parallel File System (GPFS™) and Lustre.

The I/O node includes a complete internet protocol (IP) stack, with Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) services. A subset of these services is available to user processes running on the compute nodes that are associated with an I/O node. Application processes communicate with processes running on other systems using client-side sockets. Support for server-side sockets is also provided.

The I/O node kernel is designed to be booted as infrequently as possible. The bootstrap process involves loading a ramdisk image and booting the Linux kernel. The ramdisk image is extracted to provide the initial file system. This system contains minimal commands to mount the file system on the service node using the Network File System (NFS). The boot continues by running startup scripts from NFS. It also runs customer-supplied startup scripts to perform site-specific actions, such as logging configuration and mounting high-performance file systems.

Toolchain shared libraries and all of the basic Linux text and shell utilities are local to the ramdisk. Packages, such as GPFS and customer-provided scripts, are NFS mounted for administrative convenience.

A complete description of the I/O node software is provided in Chapter 4., “Configuring I/O nodes” on page 41.

1.2.3 Application development and debugging tools

Application developers access front end nodes to compile and debug applications, submit Blue Gene/Q jobs, and perform other interactive activities.

The compilers in the Blue Gene/Q toolchain are based on the GNU compilers. The IBM XL compilers for Blue Gene/Q can also be used to build applications that run on the Blue Gene/Q compute nodes. The IBM XL compilers can provide higher levels of optimization than the Blue Gene/Q toolchain compilers because they take full advantage of the features available with the IBM Blue Gene/Q PowerPC A2 core processors.

The Blue Gene/Q system includes support for running GNU Project Debugger (GDB) with applications that run on compute nodes. Other third-party debuggers are also available.

Blue Gene/Q applications can be run in several ways. The most common method is to use a job scheduler that supports the Blue Gene/Q system, such as the IBM LoadLeveler® scheduler. Another less common option is to use the **runjob** command directly. All Blue Gene/Q job schedulers use the **runjob** interface for job submission, but schedulers can wrapper it with another command or job submission interface. The **runjob** command is described in 6.2, “The runjob command” on page 76.

See the *IBM System Blue Gene Solution: Blue Gene/Q Application Development*, SG24-7948 Redbooks publication for a complete description about how to develop applications for the Blue Gene/Q system.

1.2.4 System administration and management

The responsibilities of a Blue Gene/Q system administrator can be wide-ranging, but typically involves maintaining and monitoring the health of the Blue Gene/Q system. Most system administrator tasks are performed from the service node. The Navigator web application running on the service node plays an important role in helping an administrator perform their job. The rest of this book provides a comprehensive description of administering a Blue Gene/Q system including how to:

- ▶ Use the key features of Navigator
- ▶ Manage compute and I/O blocks
- ▶ Run diagnostics
- ▶ Perform service actions
- ▶ Use the console
- ▶ Handle alerts
- ▶ Manage various servers
- ▶ Submit and manage jobs
- ▶ Configure I/O nodes



Navigator

This chapter describes the Blue Gene Navigator administrative web application. You can use the Navigator to look at the state of your system, including the jobs, blocks, hardware, and environmentals. You can also run diagnostics, view the results, and perform service actions.

2.1 System setup

The Blue Gene Navigator is a web application. The user interface that is displayed in the browser is sent to the browser by a web server. The Navigator uses the Blue Gene Web Services (BGWS) to get the data that is displayed and also to perform other operations, such as user authentication, start service actions, and start diagnostics runs. The web server must be configured to fetch the Navigator files from the driver and to forward BGWS requests to the BGWS server. The web server that is typically used is the Apache HTTP Server. The following instructions document how to configure Apache so that the Navigator can be used.

The Blue Gene software ships with a sample Apache configuration file for the Navigator in `/bgsys/drivers/ppcfloor/navigator/etc/navigator.conf`. The configuration file provides the following functions:

- ▶ Requests for files in `/navigator` are served from the `/bgsys/drivers/ppcfloor/navigator/WebContent` directory.
- ▶ The default file in `/navigator` is `navigator.html`, the main Navigator page.
- ▶ A request for `/` is redirected to `/navigator`. The default page for the service node is the Navigator.
- ▶ Non-secured http requests are converted to secured https requests.

Using this configuration, users can get to the Navigator by entering `http://sn` in the browser location bar, where `sn` is the IP address of the service node.

The recommended method to configure Apache is to create a symlink in `/etc/httpd/conf.d` to the `navigator.conf` file in the driver directory by using the following command:

```
# ln -s /bgsys/drivers/ppcfloor/navigator/etc/navigator.conf /etc/httpd/conf.d/
```

Alternatively, you can copy the `navigator.conf` file into `/etc/httpd/conf.d/` and modify it as necessary. If you use this method, you might have to modify the `navigator.conf` file for the next release or if a fix changes the file.

In addition to the Navigator configuration, the web server must be configured to forward BGWS requests to the BGWS server. The Blue Gene software ships with a sample Apache configuration file for BGWS in `/bgsys/drivers/ppcfloor/bgws/etc/bgws.conf`. The configuration file causes requests in `/bg` to be forwarded to the BGWS server.

To configure Apache, the recommended way is to create a symlink in `/etc/httpd/conf.d` to the `bgws.conf` file in the driver directory using the following command:

```
# ln -s /bgsys/drivers/ppcfloor/bgws/etc/bgws.conf /etc/httpd/conf.d/
```

Alternatively, you can copy the `bgws.conf` file into `/etc/httpd/conf.d/` and modify it as necessary. If you use this method, you might have to modify the `bgws.conf` file for the next release or if a fix changes the file.

Configure the Apache server to start automatically using the following command:

```
# /sbin/chkconfig httpd on
```

If Apache is not running, start it with the command:

```
# /sbin/service httpd start
```

The Navigator is now ready for use.

2.2 Using the Navigator

To use the Navigator, point your web browser at `http://sn`, where `sn` is the name of the service node. You might get a warning saying that the site security certificate is not trusted. You must agree to accept the certificate to continue.

The Navigator is loaded into your browser, and you are prompted for a user name and password. The user is the user on the service node.

Tip: Use the Refresh button in the lower right to refresh the data on the panel. Using the browser refresh page button will cause the entire Navigator application to be reloaded.

The following browsers are supported:

- ▶ Firefox 3.6 - 8
- ▶ Safari 5.0 - 5.1
- ▶ Chrome 13 - 15
- ▶ Internet Explorer 8 and 9
- ▶ Opera 10.50 - 11.50

Note: Browser versions can change quickly, and certain browsers are released on a quick schedule. If a newer version of a browser is not listed in the preceding list, it might still work.

2.2.1 The Navigator window

The Navigator window is shown in Figure 2-1.

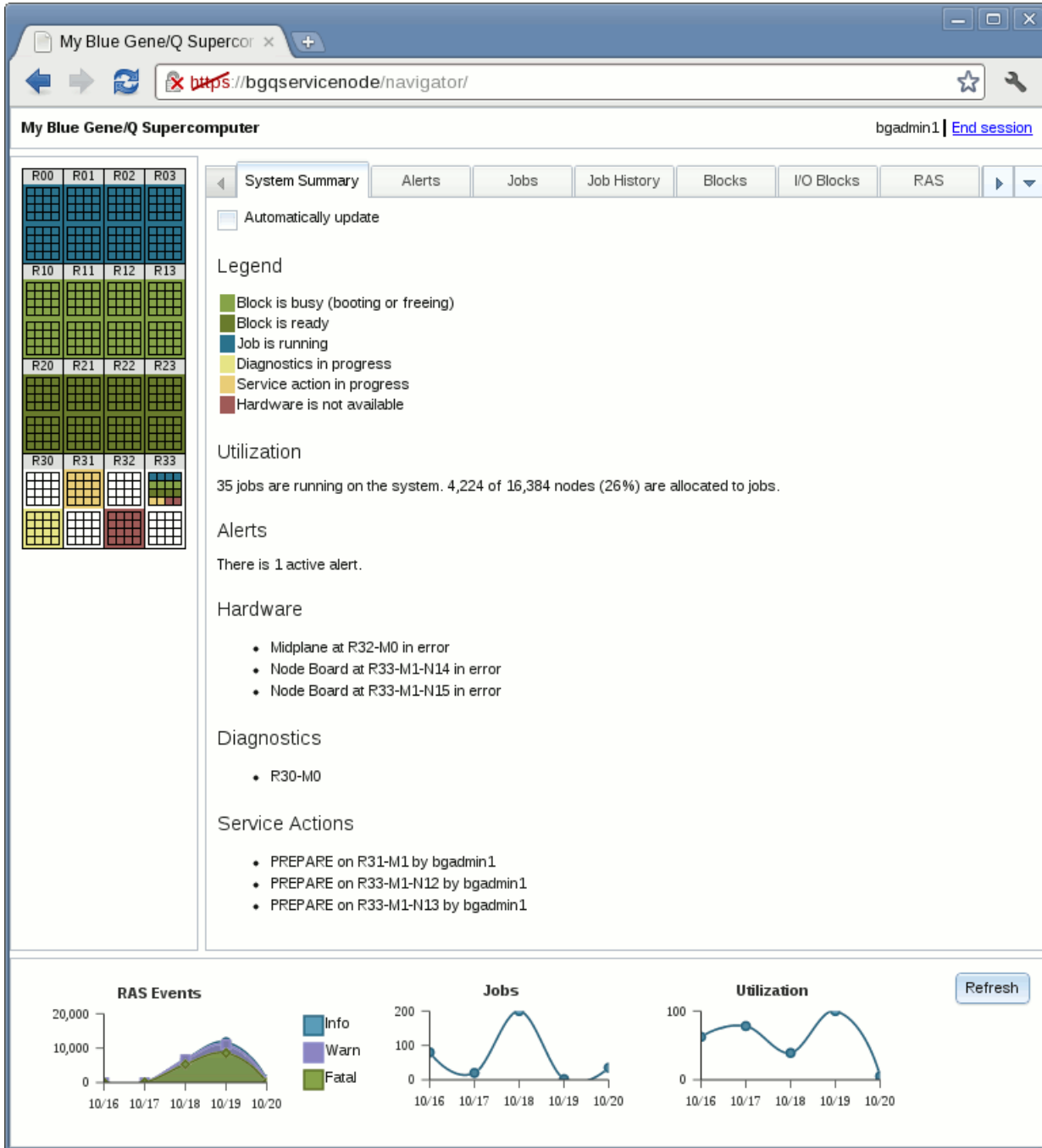


Figure 2-1 Blue Gene/Q Navigator

At the top of the Navigator page, the header provides the name of the system, the current user's name, and an End Session button. Hover the mouse pointer over your name in the header to display your special authorities, for example, administrator or hardware read. The End Session button can be used to end the current session and start a new one. An alternative way to restart a session is to use the browser reload button.

On the left is a graphical representation of the system. The hardware represented in this panel might be highlighted to display information depending on which tab is selected in the main tab area on the right. In this view, each rack is represented as a rectangle containing two midplanes, with M0 on the bottom and M1 on the top. In each midplane, the node boards are represented as squares. The upper-left square represents N00. N03 is in the upper-right corner, and N15 is in the lower-right corner.

In the footer is the dashboard, which displays several charts showing several statistics over the course of the last five days. The chart data is updated automatically every five minutes. You can click the data points in the RAS Events chart to jump to the RAS tab showing events for that day and severity. Click the Jobs chart to jump to the Job History tab.

At the lower right of the page is the Refresh button. Use the Refresh button to reload the information displayed in any of the tabs.

The following sections describe general Navigator features and each of the tabs in the main window.

2.2.2 Security integration

The Navigator is integrated with the overall Blue Gene/Q security model. For more information, see Chapter 18, “Security” on page 203. Certain pages and functions that the Navigator supports might not be available to all users depending on your authority. A user with administrative authority can view and perform all possible operations in the Navigator. A user with hardware read authority but not administrative authority can view most of the pages but cannot perform operations, such as starting a service action or starting a diagnostics run. Hover the mouse over your name in the upper-right corner of the Navigator to display what special authorities the current user has.

The description of each of the pages in the following sections includes what authority you must use on the page or for features on the page.

2.2.3 Result tables

Many of the pages in the Navigator display a table of data and also allow you to filter and sort the results. Change the sort order in the results table by clicking the header of the field that you want to sort by. Click again to sort in the opposite direction.

Tables that can be filtered have a Filter Options button that is used to set the filter for the results. Click the **Filter Options** button to display a dialog box that is used to set the values for the fields that you want to filter the results on. The filter dialog for the RAS table is shown in Figure 2-2 on page 12.

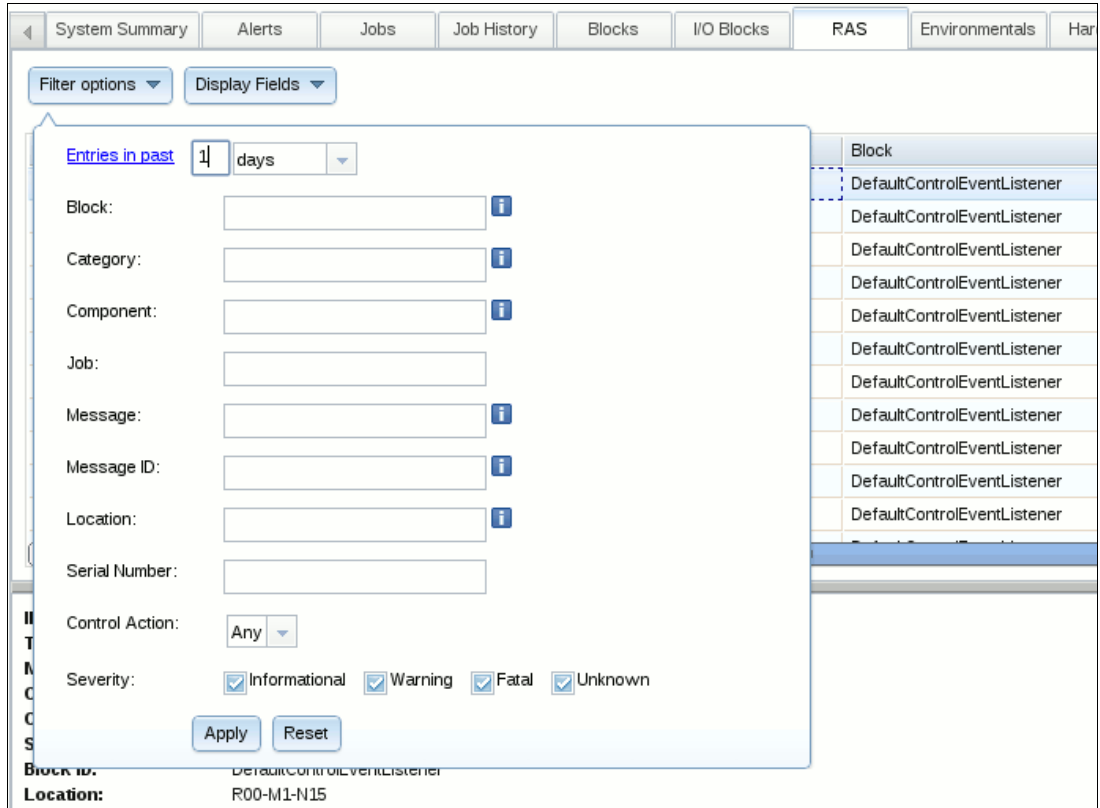


Figure 2-2 Filter options form

If a value is entered for a field, that filter is applied. Only records where the value for the field matches the value in the filter are shown in the results. The record must match all of the fields that are set. In other words, it is an AND filter rather than an OR filter.

Certain tables allow you to filter by a time interval, for example, for the past two days (a relative interval) or the records for a certain day (an absolute interval). To change the mode of the time interval filter from relative to absolute or vice-versa, click the blue text. For example, if the time interval option says “Entries in past 1 days”, click “Entries in past.” The input will change to “Entries between” so you can select an absolute time (start date and time and end date and time) rather than a relative interval. When entering an absolute time, all of the parts must be set; otherwise, the field is ignored.

Certain text fields allow a special format for the input. These fields are highlighted with an [i] icon after the input box. Hover the mouse cursor over the [i] icon, and help text describing the format of the input is displayed. For example, the Message ID field in the RAS filter options allows you to enter a pattern for message IDs. The message ID pattern allows:

- ▶ Using an * (asterisk) to match any string of characters in the value
- ▶ Entering multiple message ID patterns by separating each pattern with a space
- ▶ Filtering out message IDs by putting a minus (-) in front of the value

2.2.4 System summary

The system summary page shows a quick overview of the system. A sample screen capture for the system summary page is in Figure 2-1 on page 10. The hardware in the machine pane on the left is highlighted in several colors depending on the status of the midplane or node board. The legend in the System Summary tab shows what each color indicates:

- ▶ The hardware is used by a block that is busy booting or freeing.
- ▶ The block is ready (Initialized), but no job is running on it.
- ▶ A job is running on the block.
- ▶ Diagnostics are currently running on the hardware.
- ▶ A service action is in progress on the hardware.
- ▶ The hardware is not available because it is marked in error.

The System Summary page also shows the utilization of the system based on:

- ▶ The number of jobs
- ▶ The number of CPUs that are being used for jobs
- ▶ The total number of CPUs on the system

Details about the current jobs can be found in the Jobs tab.

The number of active alerts on the system are identified by Toolkit for Event Analysis and Logging (TEAL). Alerts can be analyzed by going to the Alerts tab.

Hardware that is not available because it was marked in error is identified in the Hardware section of the System Summary page. You can browse your Blue Gene hardware in the Hardware tab.

Any current diagnostics runs are listed in the Diagnostics section. Diagnostics details and operations are in the Diagnostics tab.

Service action operations currently in progress are listed in the Service Actions section. Details about service actions are in the Service Actions tab.

The system summary data can be refreshed automatically by selecting the **Automatically update** option at the top of the page. The data is refreshed every five seconds.

2.2.5 Alerts

The Alerts page allows you to browse and close the alerts that the Toolkit for Event Analysis and Logging (TEAL) generated, as shown in Figure 2-3.

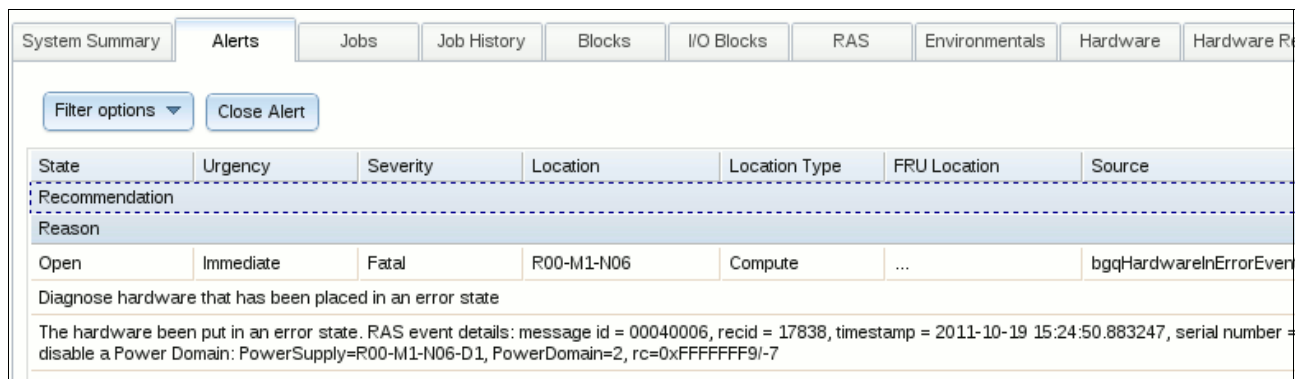


Figure 2-3 Alerts page

The initial filter options for the alert results show only open alerts that are not duplicates. To display closed alerts or duplicates, you must click the Filter options and enable the closed state or change the Duplicates option to Show.

Administrators can close alerts that are not duplicates. Select the alert, and if its state is Open, you can click the **Close Alert** button. Administrators can also remove alerts that are closed.

You must have hardware read authority; otherwise, the Alerts tab is hidden. You must have administrative authority to close an alert.

2.2.6 Jobs and job history

The Jobs page shows jobs that are currently running on the system. Select a specific job, and details for the job are displayed in the details pane at the bottom of the page. To see RAS events for the job, if there are any, switch to the RAS tab in the details pane. A screen capture of the page is shown in Figure 2-4.

Job ID	Status	User	Executable	Block
90	Running	bguser1	hello_mpi.bg	R00
91	Running	bguser1	hello_mpi.bg	R00
92	Running	bguser1	hello_mpi.bg	R00
93	Running	bguser1	hello_mpi.bg	R00
94	Running	bguser1	hello_mpi.bg	R00
95	Running	bguser1	hello_mpi.bg	R00
96	Running	bguser1	hello_mpi.bg	R00

Details		RAS	
ID:	92		
Executable:	hello_mpi.bg		
User:	bguser1		
Block:	R00		
Initial directory:	/home/bguser1		
Arguments:	"		
Environment:	"		
Started:	10/20/11 9:08 AM		
Status:	Running		
Nodes used:	256		
Shape:	4x2x4x4x2		
Processes per node:	1		
NP:	0		
Client:	myfen1.28727		

Figure 2-4 Jobs page

The Job History page is similar to the Jobs page, but it shows completed jobs that are no longer running on the system. An image of the page is shown in Figure 2-5.

The screenshot shows the Job History page with a table of job entries and a details pane for job ID 2005.

Job ID	Status	User	Executable	Block	Size	PPN	Started
2001	Terminated	bguser1	hello_mpi.bg	R00	128	8	10/16/11 12:00 AM
2002	Terminated	bguser1	hello_mpi.bg	R00	128	8	10/16/11 12:00 AM
2003	Terminated	bguser1	hello_mpi.bg	R00	128	8	10/16/11 12:00 AM
2004	Terminated	bguser1	hello_mpi.bg	R00	128	8	10/16/11 12:00 AM
2005	Terminated	bguser1	hello_mpi.bg	R00	128	8	10/16/11 12:00 AM
2006	Terminated	bguser1	hello_mpi.bg	R00	128	8	10/16/11 12:00 AM
2007	Terminated	bguser1	hello_mpi.bg	R00	128	8	10/16/11 12:00 AM
2008	Terminated	bguser1	hello_mpi.bg	R00	128	8	10/16/11 12:00 AM

Details		RAS
ID:	2005	
Executable:	hello_mpi.bg	
User:	bguser1	
Block:	R00	
Initial directory:	/home/bguser1	
Arguments:	"	
Environment:	"	
Started:	10/16/11 12:00 AM	
Ended:	10/17/11 12:00 AM	
Status:	Terminated	
Nodes used:	128	
Shape:	2x2x4x4x2	
Processes per node:	8	
NP:	0	
Client:	myfen1:1	
Exit status:	0	

Figure 2-5 Job history page

Users must have read authority to the job or they will not see it in the jobs or job history pages.

2.2.7 Blocks and I/O blocks

The Blocks page shows the compute blocks that are defined on the system and allows users to create and delete blocks. When a block is selected, the hardware that it uses is highlighted in the machine view. If the midplane is included in the block, it is highlighted in blue. However, if a midplane is passed through, it is highlighted in orange. To show details for a block, select the block. You can see the jobs that are running or have run on the block by switching to the Jobs tab in the details pane. A screen capture of the Blocks page is shown in Figure 2-6 on page 16.

To define a new block, press the **Create** button to cause the Block Builder page to be displayed. For information about the Block Builder, see Section 2.2.16, "Creating blocks with the Block Builder" on page 27.

To delete a block, select the block and click **Delete**. The status of the block must be *Free* for it to be deleted.

Block ID	Status	User	Size
R00	Initialized	bknudson	4,096
R10	Booting	bknudson	4,096
R20	Initialized	bknudson	4,096
R331-00-03	Initialized	bknudson	128
R331-04-07	Booting	bknudson	128
R331-08-4	Initialized	bknudson	128
_DIAGS_R30-M0	Initialized	bknudson	512

Details	
ID:	R10
Status:	Booting
Status Changed:	10/20/11 9:08 AM
User:	bknudson
Owner:	bknudson
Size:	4,096
Shape:	8x4x8x8x2
Torus:	ABCDE
Description:	Generated via web services
Created:	10/20/11 9:08 AM

Figure 2-6 Blocks page

The I/O Blocks page, Figure 2-7, shows the I/O blocks.

Block ID	Status	User	Size
IO	Initialized	bgadmin1	128

ID:	IO
Status:	Initialized
Status Changed:	10/20/11 10:26 AM
User:	bgadmin1
Owner:	bgadmin1
I/O Nodes:	128
Locations:	R00-ID, R01-ID, R02-ID, R03-ID, R10-ID, R11-ID, R12-ID, R13-ID, R20-ID, R21-ID, R22-ID, R23-ID, R30-ID, R31-ID, R
Description:	Generated via genloBlock
Created:	10/20/11 10:26 AM

Figure 2-7 I/O blocks

You must have:

- ▶ Read authority to the block to display the block
- ▶ Create authority to create a block
- ▶ Write authority to the block to delete the block

2.2.8 RAS

The RAS page, shown in Figure 2-8 on page 18, allows you to query the RAS events that have occurred on the system. To view details for a RAS event, click the RAS event in the table.

The results table on the RAS page has an extra feature where you can choose which fields you want displayed. Click the **Display Fields** button, and select only those fields that you want displayed.

System Summary	Alerts	Jobs	Job History	Blocks	I/O Blocks	RAS	Environmentals	Hardware	Hardware Replace
----------------	--------	------	-------------	--------	------------	------------	----------------	----------	------------------

Filter options ▾	Display Fields ▾
------------------	------------------

Record ID	Time	Severity	Category	Message ID	Block	Location
19712	2011-10-20 09:51:42.000000	INFO	BQC	00070218	DefaultControlEventListener	R00-M1-N01
19710	2011-10-20 09:49:07.000000	INFO	Card	00070207	DefaultControlEventListener	R00-M0-N06
19709	2011-10-20 09:48:39.000000	INFO	Card	0007020F	DefaultControlEventListener	R00-M0-N06
19708	2011-10-20 09:48:47.685647	INFO	Card	0004005B	DefaultControlEventListener	R00-M0-N06
19707	2011-10-20 09:30:23.000000	INFO	Card	0007020A	DefaultControlEventListener	R00-M0-N06
19706	2011-10-20 09:30:31.171274	INFO	Card	000400B7	DefaultControlEventListener	R00-M0-N06

ID:	19712
Time:	2011-10-20 09:51:42.000000
Message ID:	00070218
Category:	BQC
Component:	BAREMETAL
Severity:	INFO
Block ID:	DefaultControlEventListener
Location:	R00-M1-N01
Message:	Cable from R00-M1-N01-T09 to R00-M0-N01-T11 on NodeBoard R00-M1-N01 contains bad wires. Cable is good but has 1 broken wires (0
Description:	VerifyCables detected bad wires. Examine the VerifyCables log file for detailed information.
Service Action:	Run VerifyCables on this card

Figure 2-8 RAS

You must have hardware read authority; otherwise, the RAS tab will be hidden.

2.2.9 Environmentals

The Blue Gene/Q system collects various environmental data while it is running. The Environmentals page, shown in Figure 2-9 on page 19, allows you to query the environmental data. Each tab on the Environmentals page shows a different aspect of the environmental data that has been collected.

Location	Time	ASICTemp
R00-M0-N00-J19	10/20/11 10:34 AM	48
R00-M0-N00-J20	10/20/11 10:34 AM	45
R00-M0-N00-J21	10/20/11 10:34 AM	48
R00-M0-N00-J22	10/20/11 10:34 AM	45
R00-M0-N00-J23	10/20/11 10:34 AM	45
R00-M0-N00-J24	10/20/11 10:34 AM	44
R00-M0-N00-J25	10/20/11 10:34 AM	46
R00-M0-N00-J26	10/20/11 10:34 AM	45
R00-M0-N00-J27	10/20/11 10:34 AM	46
R00-M0-N00-J28	10/20/11 10:34 AM	42
R00-M0-N00-J29	10/20/11 10:34 AM	46
R00-M0-N00-J30	10/20/11 10:34 AM	50
R00-M0-N00-J31	10/20/11 10:34 AM	43
R00-M0-N01-J00	10/20/11 10:34 AM	25
R00-M0-N01-J01	10/20/11 10:34 AM	26
R00-M0-N01-J02	10/20/11 10:34 AM	26
R00-M0-N01-J03	10/20/11 10:34 AM	27

Figure 2-9 Environmentals

You must have hardware read authority or the Environmentals tab is hidden.

2.2.10 Hardware

The Hardware page provides a way to browse the Blue Gene/Q hardware. Hardware that is not available is highlighted in red in the machine view when viewing this page.

A summary of the machine is initially displayed. Display details for a rack by clicking the rack on the machine view. If rack details are being displayed, click a midplane in the rack to display details for the midplane. If a midplane is displayed, click a node board in the midplane to display details for a node board. The upper-left square is the N00 node board, the upper-right square represents N03, and the lower-right square represents N15. If the details for a piece of hardware contain a link to another piece of hardware, click the location highlighted in blue to jump to that location.

The Parent button, shown in Figure 2-10 on page 20, displays the piece of hardware that the currently selected hardware is in. For example, pressing the Parent button when a node board is selected switches to the midplane that it is contained in.

Another way to display details for a piece of hardware is the Jump To input box. Enter a hardware location to jump directly to details for the hardware at that location. For example, jumping to R00-M0-N00 jumps directly to the node board at R00-M0-N00.

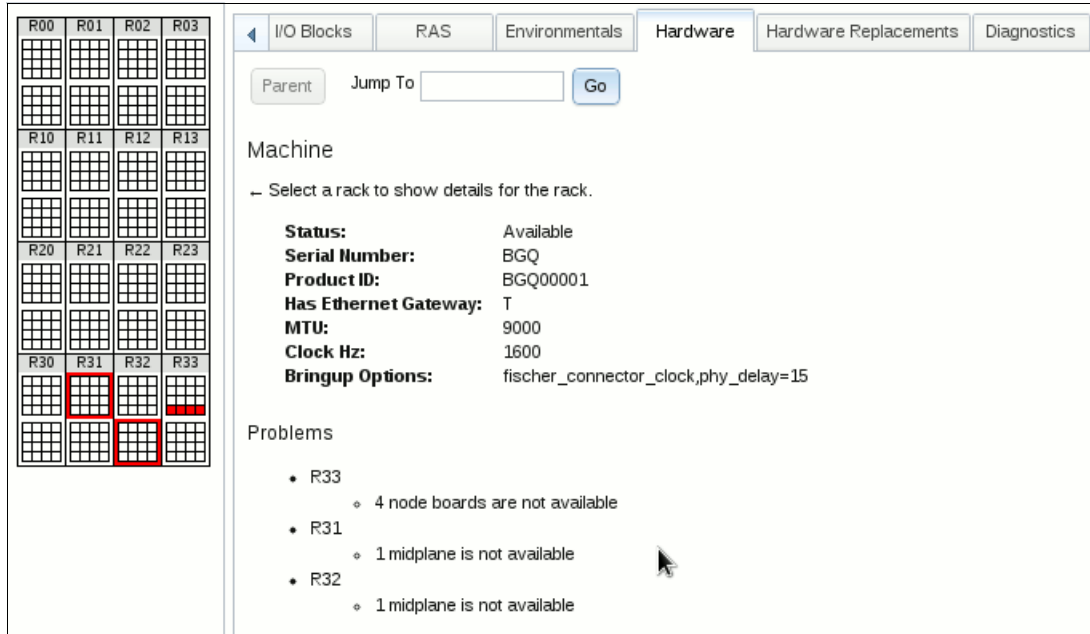


Figure 2-10 Hardware page

You must have hardware read authority or the Hardware tab is hidden.

2.2.11 Hardware replacements

The Hardware Replacements page shows the history of hardware that was replaced on the system. You can filter the results by time, hardware type, location, serial number, or ECID. A screen capture of the Hardware Replacements page is shown in Figure 2-11.

Time	Type	Location	Old Serial Number	New Serial Number	Old ECID
9/22/11 8:46 AM	Node	R00-M0-N09-J12	46K4894YL30K0222044	46K4894YL30K100500C	018524005715070E1812C4B3B
9/20/11 8:38 AM	Node	R10-M0-N07-J27	46K4894YL30K022202C	46K4894YL30K1005019	01852400541D770E1818C6B27
9/7/11 10:00 AM	Node	R10-M0-N05-J03	46K4897YL30K0223015	46K4897YL30K0365017	018524005709E70E081BC5B23

Figure 2-11 Hardware replacements

You must have hardware read authority or the Hardware Replacements tab is hidden.

2.2.12 Diagnostics

The Diagnostics page, shown in Figure 2-12, allows you to view current and completed diagnostics runs and also to start a new run.

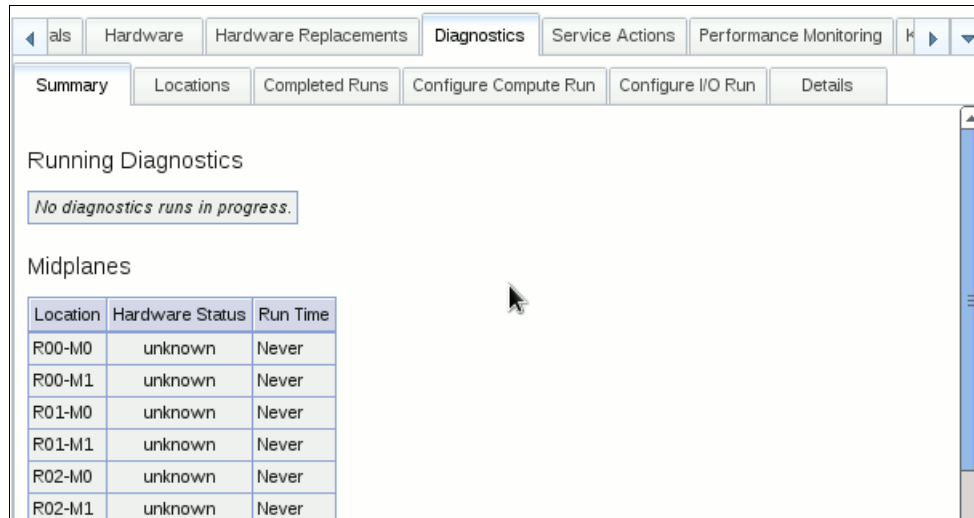


Figure 2-12 Diagnostics page

The results of current and completed diagnostics runs can be queried from different perspectives. The Summary tab provides a summary of running diagnostics and the results of diagnostics runs on midplanes, I/O drawers, and user-defined blocks. Click a current run, a midplane location, or an I/O drawer location to display the results of diagnostics on that midplane or I/O drawer. The results are displayed in the Details tab.

The Locations tab shows the diagnostics results at all of the locations where diagnostic tests provided a result. The location might be a node board, node card, I/O drawer, and so on. Click a row in the results table to display detailed results for that location. The initial result filter shows the latest result for each piece of hardware. Change the filter to a location and set the time interval to display the results for the hardware at the location for various runs in the specified interval.

The Completed Runs tab provides the capability to query the diagnostics runs performed on the system. Click a row in the result column to display details for that run.

Administrators can configure and submit a new diagnostics run from the Navigator using either the Configure Compute Run tab, Figure 2-13 on page 22, or the Configure I/O Run tab. To configure a diagnostics run on a midplane, first select the midplane or midplanes that you want to run diagnostics on. Next, you can check or uncheck the diagnostics buckets of test cases that you want to run. An estimate of the run time is displayed next to the bucket. Changing which buckets are selected causes the test cases that are in all the selected buckets to be checked or unchecked in the table of test cases.

You can optionally override the bucket selection and pick individual test cases by selecting test cases from the table of test cases. The estimated run time is updated based on the selected test cases. Select the run options that you want to apply to the test. The run options are described in Chapter 10, “Diagnostics” on page 119.

When you are ready to start the diagnostics run, click the **Start Diagnostics Run** button and the diagnostics script is started on the service node. After a run ID is generated by the script,

it will be displayed in the Navigator. You can view the current status and progress of the run by going to the Summary tab.

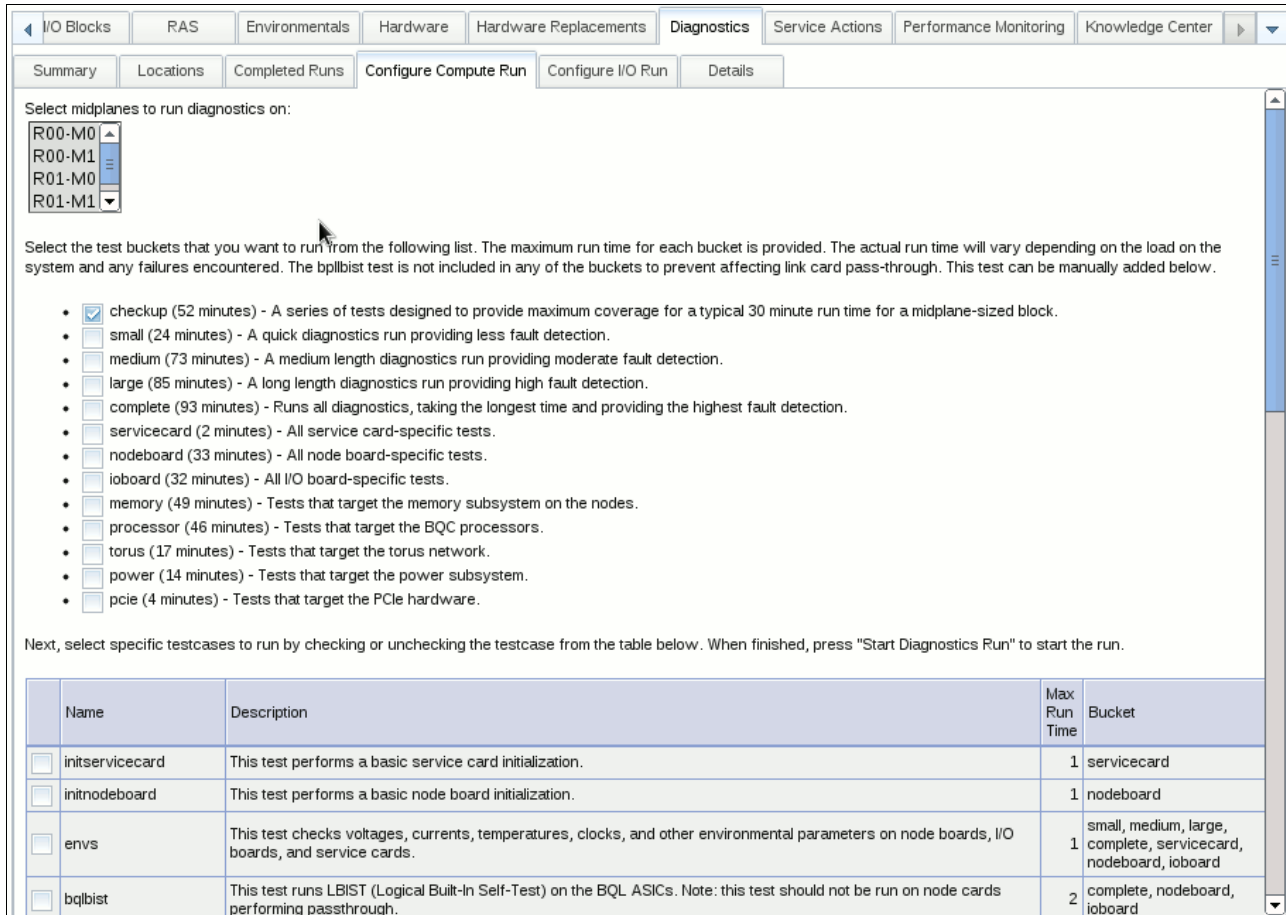


Figure 2-13 Configure diagnostics run

You must have hardware read authority to view the Diagnostics page and administrative authority to start a diagnostics run.

2.2.13 Service actions

You can view the status of service actions, either in progress or completed, prepare hardware for service, and end service actions in the Service Actions page, shown in Figure 2-14, of the Navigator.

ID	Location	Action	Status	Prepared By	Prepared	Ended By	Ended
23	R33-M1-N13	PREPARE	Prepared	bgadmin1	10/20/11 9:08 AM
22	R33-M1-N12	PREPARE	Prepared	bgadmin1	10/20/11 9:08 AM
21	R31-M1	PREPARE	Prepared	bgadmin1	10/20/11 9:08 AM

Figure 2-14 Service actions

The default filter for the service actions results shows all service actions that were started in the past week. You can filter the service actions results by action, status, location, user, and so on.

You can end a service action from this page. Select a service action from the table that has an Action of PREPARE and a Status of Prepared, and click the **End Service Action** button.

Administrators can prepare hardware for service by clicking the **Prepare Hardware for Service** button on the Service Actions page. This action opens a new tab in the main tabs to a wizard to help you pick the type of service action and the hardware. A screen capture of the initial Prepare Service Action page is shown in Figure 2-15.

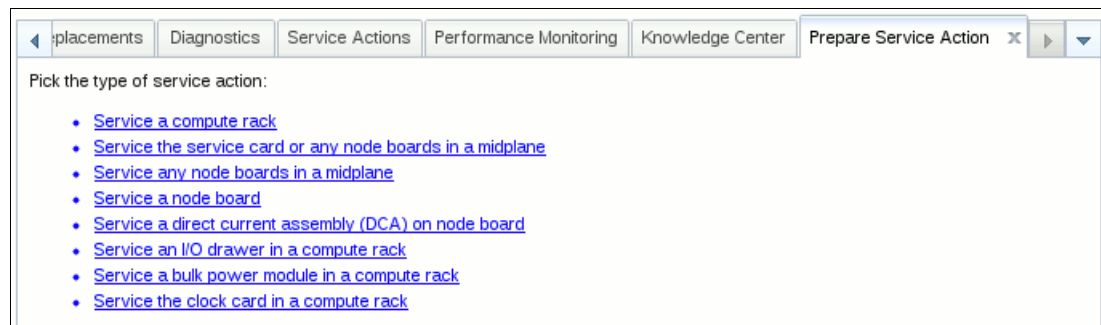


Figure 2-15 Prepare service action

Select the type of service action that you want to perform by clicking the link. The type of service action to pick based on the service you need to perform can be found in Chapter 9, “Service actions” on page 107.

After you select the type of service action, a page is displayed that allows you to pick the hardware location. It also shows the jobs that are affected if the service action is started. A screen capture of this page for a service action on a midplane is shown in Figure 2-16. Click the **Start Prepare for Service** button when you are ready to start the service action. The Navigator will show the service action ID after it is started, and you can check the progress on the Service Actions page. To pick a different type of service action, click the **Back** button. When you are done with this page you can close the tab using the **X** button on the tab header.

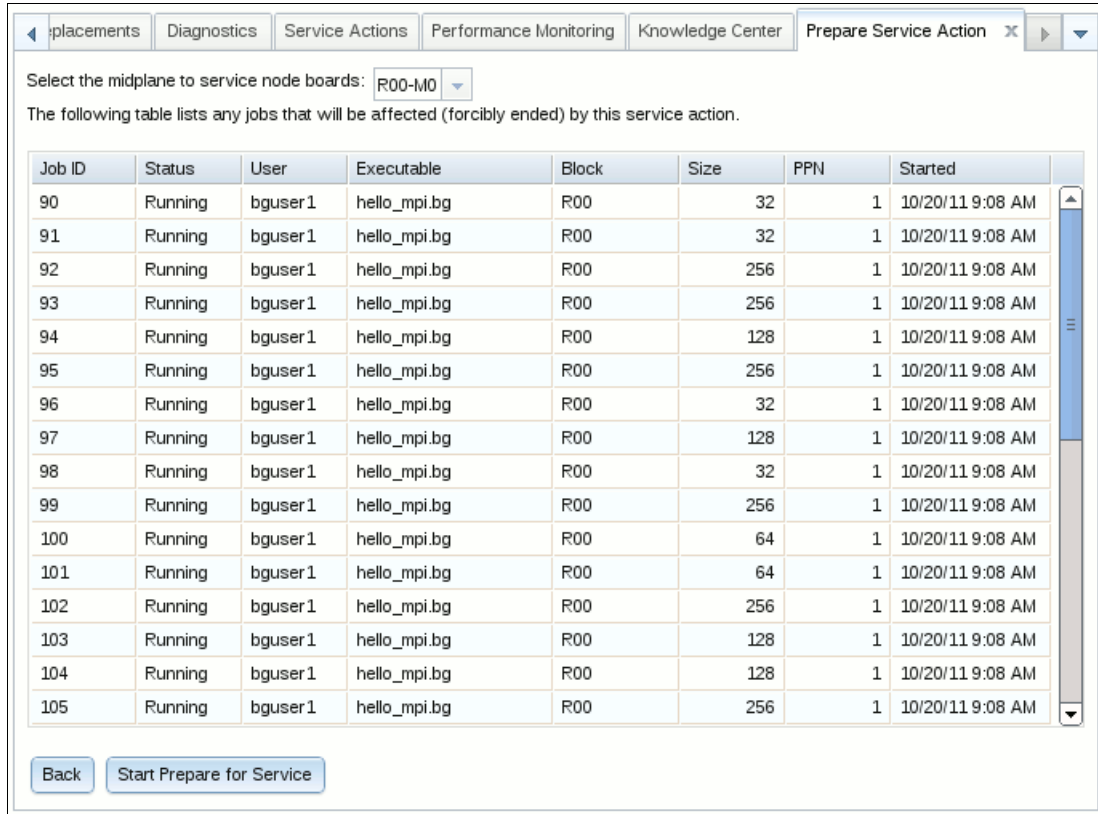


Figure 2-16 Prepare midplane for service

You must have hardware read authority to view the Service Actions page and must be an administrator to prepare hardware for service or to end a service action.

2.2.14 Performance monitoring

Using the Performance Monitoring page, shown in Figure 2-17 on page 25, you can examine the block boot performance data that is captured while the machine is running. Each time a block is booted several durations are collected at each step of the boot to provide an overall boot time. The boot performance might be affected by changes to the software and by environmental changes, such as disk or network performance.

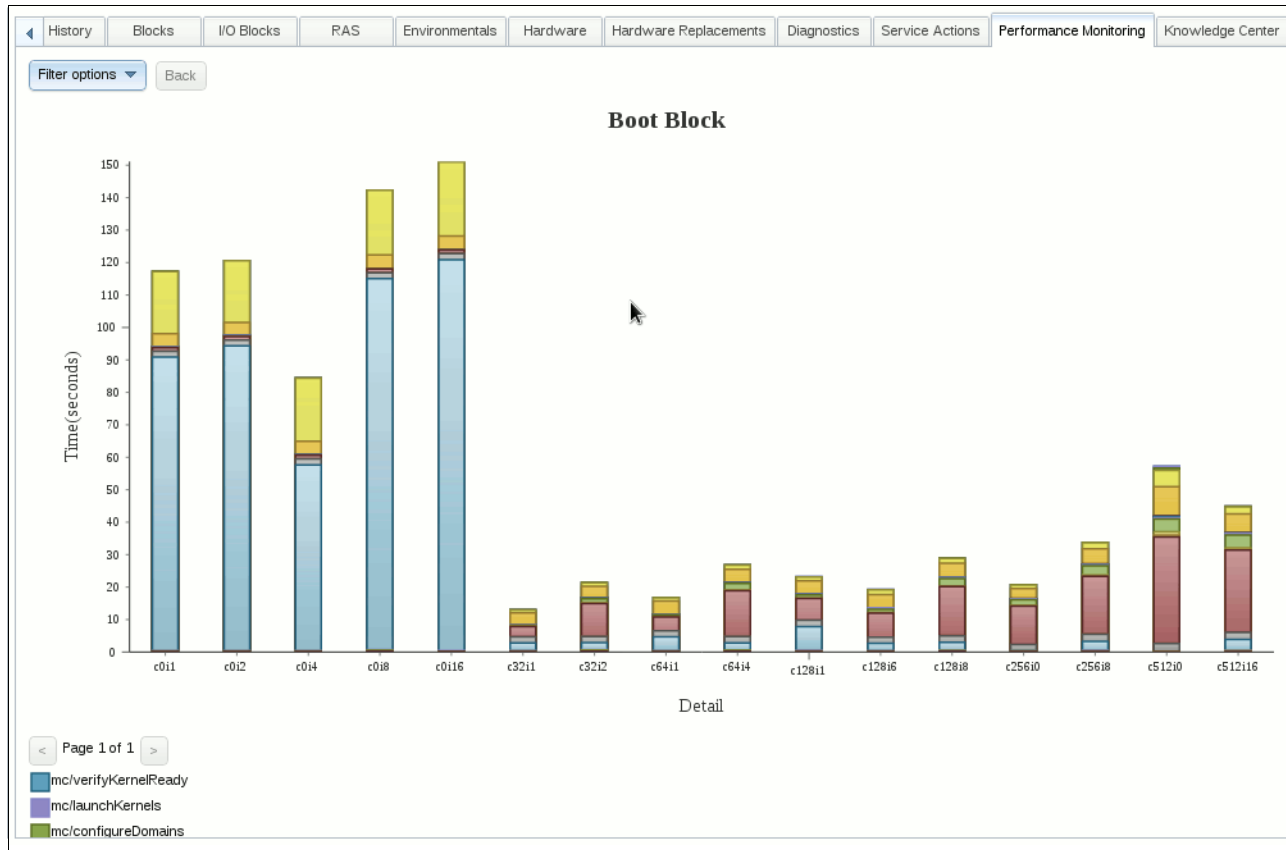


Figure 2-17 Performance monitoring

The boot block performance data uses the “detail” to discriminate among block boots that are similar. Block boot performance is typically a function of the size of the block (the number of compute nodes) and the amount of I/O available (I/O nodes). The detail in the boot block performance measurements, as displayed in the Navigator, are a concatenation of the number of compute nodes and number of I/O nodes, such as “c<compute nodes>i<i/o nodes>”. For example, a boot of a rack-sized compute block connected to eight I/O nodes has a detail of “c1024i8”.

Hover the mouse cursor over each section of the bars in the chart to get details for that section, such as the name of the step and the average duration. Click a bar in the chart to change the view to show the result for that detail using a pie chart.

The default filter shows the results grouped by detail for the past seven days. You can set the filter option to display a specific detail or block ID, the time interval for the measurements, and select how the results are grouped (daily, weekly, or monthly). The chart changes depending on the filter options. If the filter has a detail or block ID set and the grouping is not set, a pie chart is displayed, similar to Figure 2-18 on page 26.

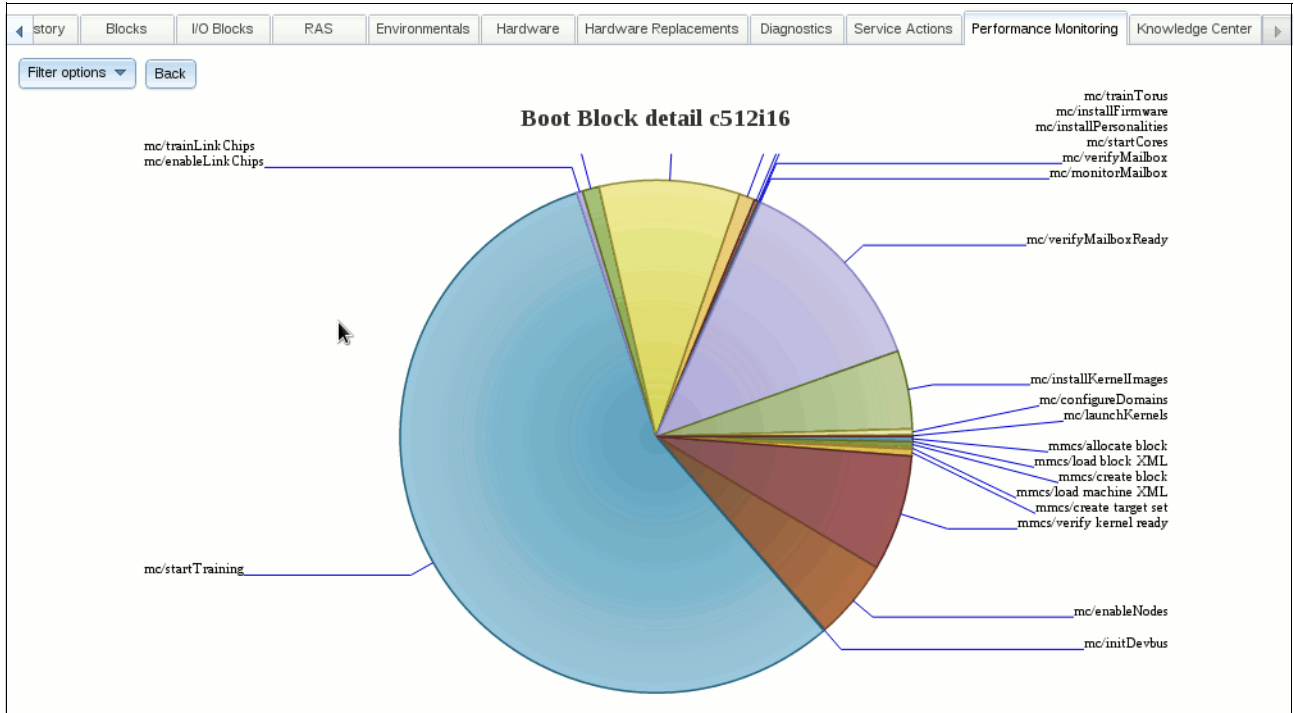


Figure 2-18 Performance monitoring details

When the filter has a detail or block ID and the time interval grouping is set to something other than “None”, a bar chart is displayed. The results are grouped by day, week, or month. Figure 2-19 shows sample output for the detail “c256i8” grouped by day.

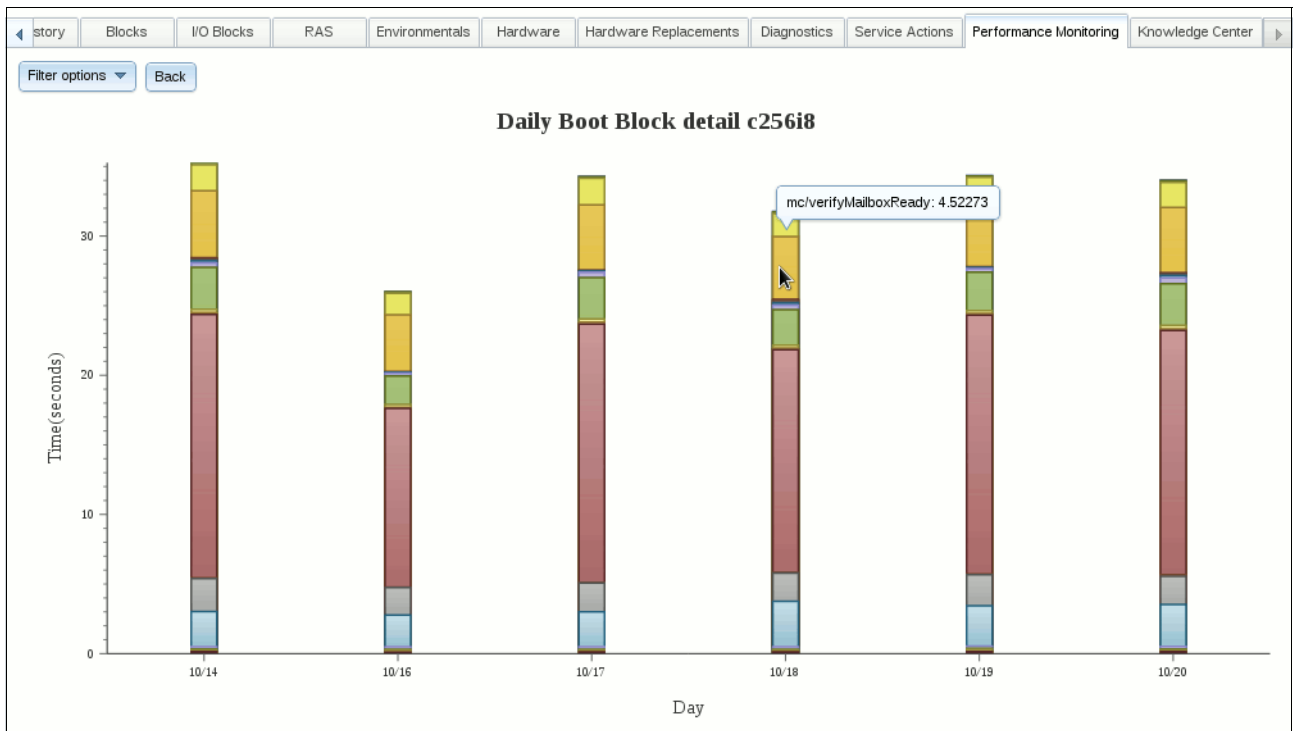


Figure 2-19 Performance monitoring grouped by day

You must have hardware read authority to view the Performance Monitoring tab.

2.2.15 Knowledge Center

The Knowledge Center contains links to online sources of documentation on the Blue Gene/Q system, as shown in Figure 2-20. Click the link to open the documentation in your browser.

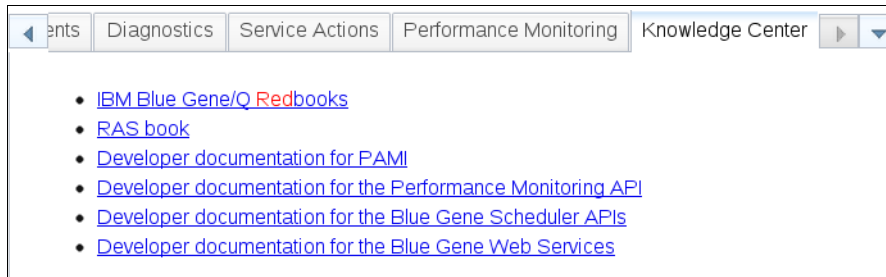


Figure 2-20 Knowledge Center

Any user can view the Knowledge Center page.

2.2.16 Creating blocks with the Block Builder

When you click the **Create Block** button on the Blocks page, the Block Builder is displayed in a new tab. General information about blocks, including knowing how the midplanes are connected in a torus and using the command-line tools to create blocks, is described in Chapter 3, “Compute and I/O blocks” on page 29. A screen capture of the Block Builder is shown in Figure 2-21

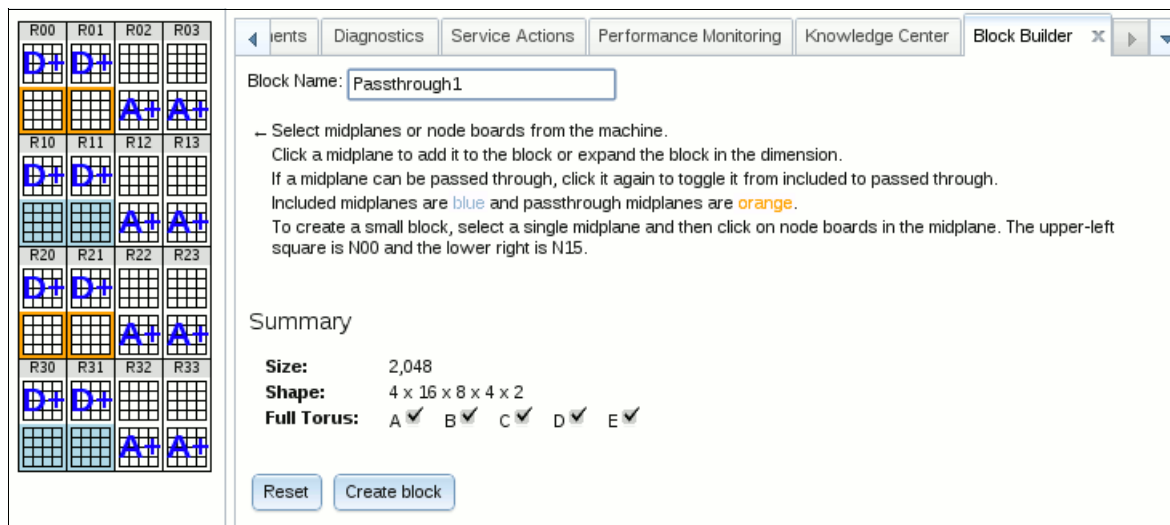


Figure 2-21 Navigator block builder

You create a block by selecting midplanes or node boards from the machine view on the left. To create a large block, click a midplane. The midplane changes color to blue to indicate that it is included. The next midplane in each dimension according to the torus cabling is annotated with the dimension. Clicking one of these annotated midplanes expands the block in that dimension. You can continue in this manner until the block is defined. To set a midplane to be passed through, click the midplane again to toggle it to pass through. Pass-through midplanes are highlighted in orange. The block must have three or more included midplanes in the dimension for a midplane to be toggled to pass through.

To create a small block, first click a midplane. Next, click the smaller squares representing node boards in that midplane to select the node boards in the small block. The upper-left corner is N00, the upper-right corner is N03, and the lower-right corner is N15. The Block Builder does not allow you to create a small block that is not valid, so valid combinations of node boards must be selected.

The Navigator displays a summary of the block that is under construction. The size is the number of compute nodes in the block, 512 for each midplane or 32 for each node board. The shape shows the shape in each dimension (A x B x C x D x E) for the block. The “Full Torus” indicators show whether the block is a torus in each dimension. A large block is a torus in the dimension if the number of midplanes in the dimension is either 1 or the size of the machine in that dimension. For a small block, the connectivity in each dimension can be determined from the size.

After a name is assigned and the hardware is selected, press the Create Block button to create the block.

You must have block create authority to use the Block Builder.



3

Compute and I/O blocks

On Blue Gene/Q, there are two types of blocks: *compute blocks* and *I/O blocks*. Compute blocks identify the hardware on Blue Gene/Q on which jobs run. I/O blocks allow the compute blocks to communicate with resources outside of the compute nodes. I/O nodes are not tightly coupled with the compute nodes they serve. Instead, compute nodes and I/O nodes are separate entities, connected through I/O links.

3.1 Compute blocks

Compute blocks consist of a number of compute nodes. Before you can submit any job to Blue Gene/Q compute nodes, you must create a block that defines the resources that are used for the job. Each block definition is identified by a unique *block ID*. The Navigator Blocks page (see section 2.2.7, “Blocks and I/O blocks” on page 15) shows the compute blocks that are defined on the system. The hardware and cabling of the Blue Gene/Q system determines which blocks you can create on your system.

The Blue Gene/Q system contains one or more *midplanes*. Midplanes are cabled together using link chips 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/Q system is cabled. You can also subdivide midplanes 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 of different sizes, such as 32, 64, 128, or 256 compute nodes. The size of small blocks that you can create depends on the number of I/O nodes that your system has. Each block needs at least one connected I/O node. On a system with the maximum number of I/O nodes, you can subdivide a midplane into 16 blocks, each with 32 compute nodes.

A subset of the compute nodes are connected to I/O links that connect to I/O nodes. The I/O-linked compute nodes route I/O traffic to the I/O block. Compute blocks are booted and attached to an already booted I/O block.

Compute blocks can be either *dynamic* or *static*. Dynamic blocks are created using a job scheduler, such as LoadLeveler. The user that submits the job tells the job scheduler details about the Blue Gene/Q resources, such as the number of processors, that are needed for the job. The job scheduler then creates the block based on this information using the Scheduler APIs after examining which Blue Gene/Q resources are available at the time to run the job. The user cannot choose which resources are used. When the job completes the block is freed and deleted. The system administrator creates static blocks, and they remain available to be reused when needed. These static blocks allow the system administrator to assign a block for a particular user to use specific compute resources. When users submit their jobs, they specify only which static block they are going to use to execute the job.

3.1.1 Large blocks

Large blocks consist of all the compute nodes in one or more complete midplanes. They are always a multiple of 512 compute nodes and can be a torus in five dimensions. Large blocks can be created in several ways: on the Blocks page in Navigator, by job schedulers that use Blue Gene/Q Scheduler APIs or using various MMCS block creation commands in `bg_console`.

Figure 3-1 on page 31 is an abstract view of a 192 midplane system. Each box in the figure represents a single midplane. A midplane, which consists of 512 compute nodes, is referred to as a 4x4x4x4x2 torus. The Blue Gene/Q hardware provides a five-dimensional torus. The dimensions are denoted by the letters A, B, C, D, and E. Figure 3-1 on page 31 shows only four dimensions because the E dimension is always 2, and is contained entirely within a midplane. Therefore, for the purposes of multi-midplane, large blocks, the midplanes are combined in four dimensions. Every block on Blue Gene/Q, large or small, has an E dimension of size 2. When multiple midplanes are cabled together, a torus direction that has at least two midplanes existing in one rack is the D dimension. When cables go down a row of racks, the direction is the C dimension. When cables go down a column of racks, the direction is the B dimension. The remaining direction, which can go down a row or column (or both), is

the A dimension. When two sets of cables go down a row or column, the longest cables define the A dimension.

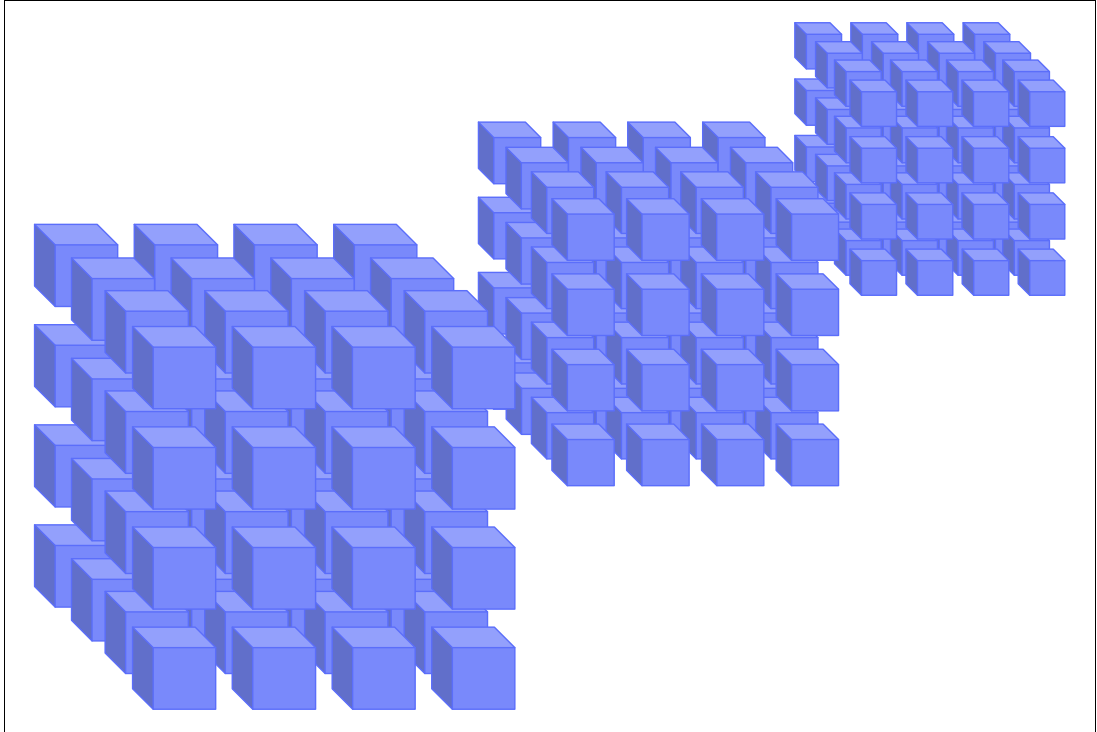


Figure 3-1 Large block

The physical connection between midplanes is achieved by using a hardware component called Blue Gene/Q link chips. Blue Gene/Q link chips might also be referred to as BQL chips. These link chips reside on the node boards that make up a midplane. To make a torus connection between two midplanes, there are 16 physical cables used, one for each node board. The link chips have settings to control how the midplanes are connected. These settings are made by the Control System software when a block is booted. There are four possible configurations that link chips can have:

- ▶ Wrap midplane
- ▶ Include midplane with both input/output ports enabled
- ▶ Include midplane with output port enabled and input port disabled (set for blocks with mesh dimensions)
- ▶ Include midplane with input port enabled and output port disabled (set for blocks with mesh dimensions)

Figure 3-2 on page 32 shows a representation of those settings. If the link chip is set to "wrap", the midplane is not included in a multiple midplane block in that dimension. However, the link chips might still be used by another block to pass through the midplane. When the link chips are set to "include", the midplane takes part in a multiple midplane block, in that dimension.

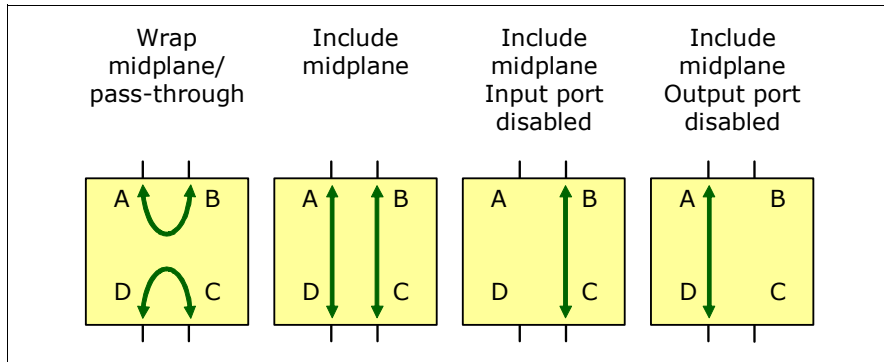


Figure 3-2 Link chip settings

The link chips are set independently for each dimension. A link chip set to wrap for the A dimension ensures that the midplane is not connected to other midplanes in the A dimension. However, it can still have the B, C, and D dimension link chips set to include, and therefore be part of a multi-midplane block.

Figure 3-3 shows the link chip settings for several different blocks. The maximum number of midplanes in any given direction is four.

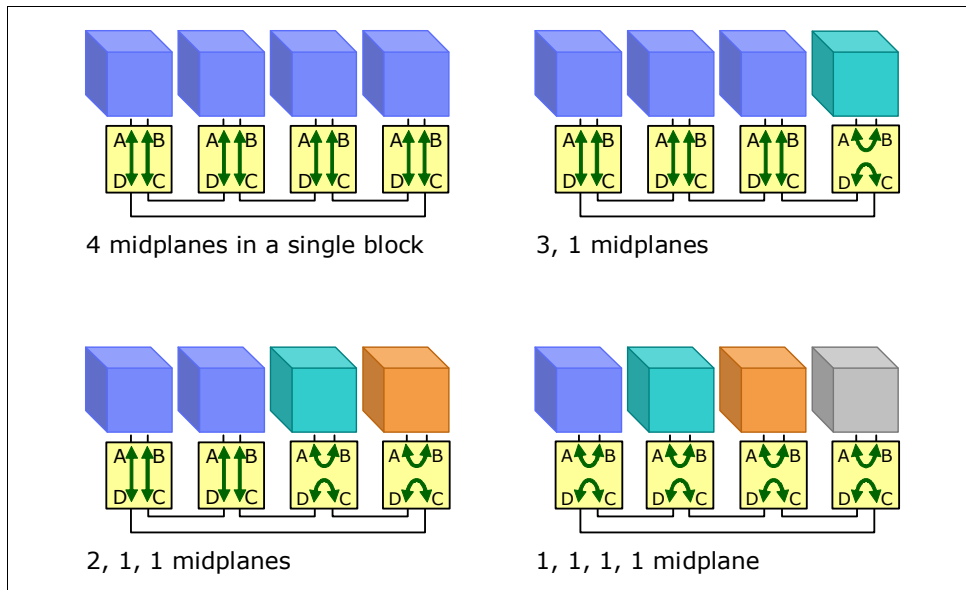


Figure 3-3 Possible link settings

The figures that follow show a few of the possible large block configurations available in a large system environment.

Figure 3-4 shows three 4x4x4x1 blocks. Blue Gene/Q nomenclature uses A, B, C, D, and E to denote the five dimensions of a block. The last dimension, E, is fixed at two and is not a factor when creating blocks.

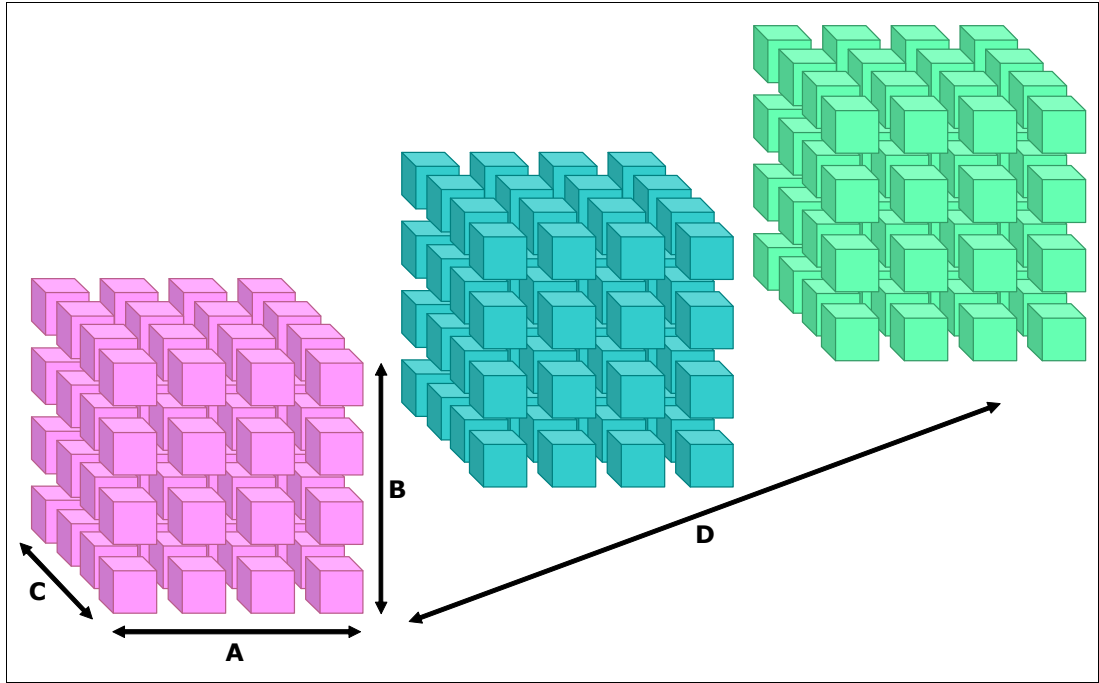


Figure 3-4 Block dimensions

Figure 3-5 shows four 4x4x1x1 blocks (on the left) and two 4x4x4x1 blocks.

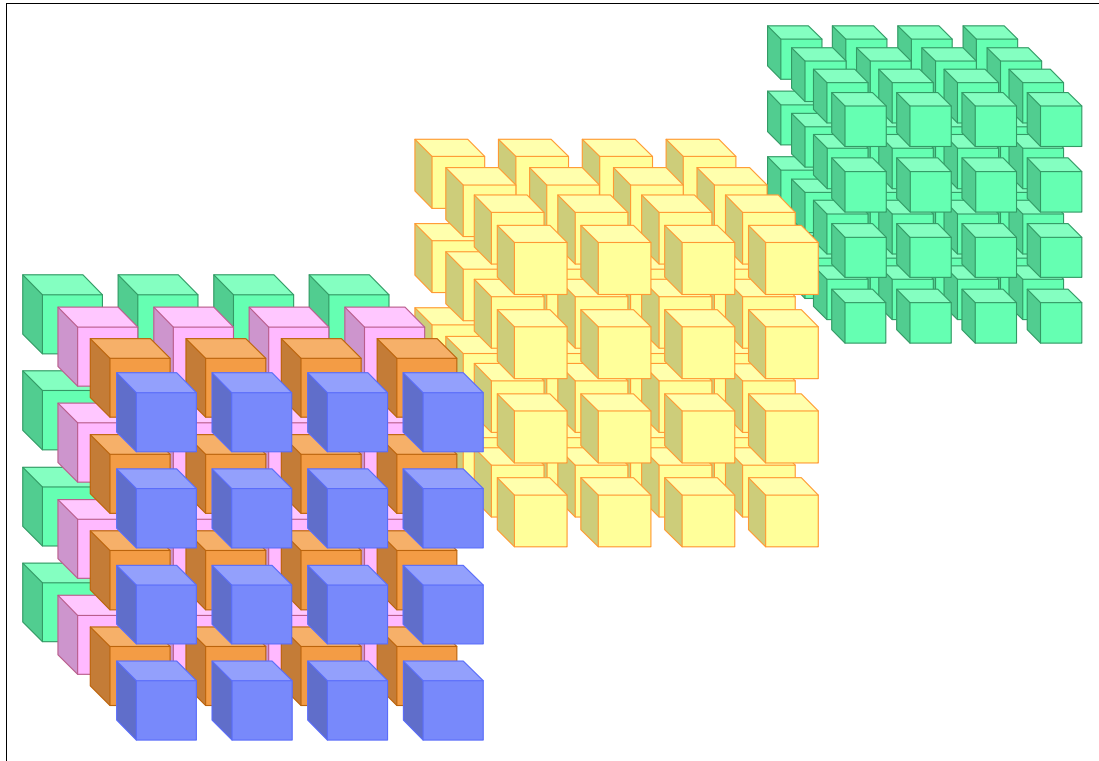


Figure 3-5 Two 64 midplane blocks and four 16 midplane blocks

Figure 3-6 shows two blocks each with four midplanes and one 8-midplane block.

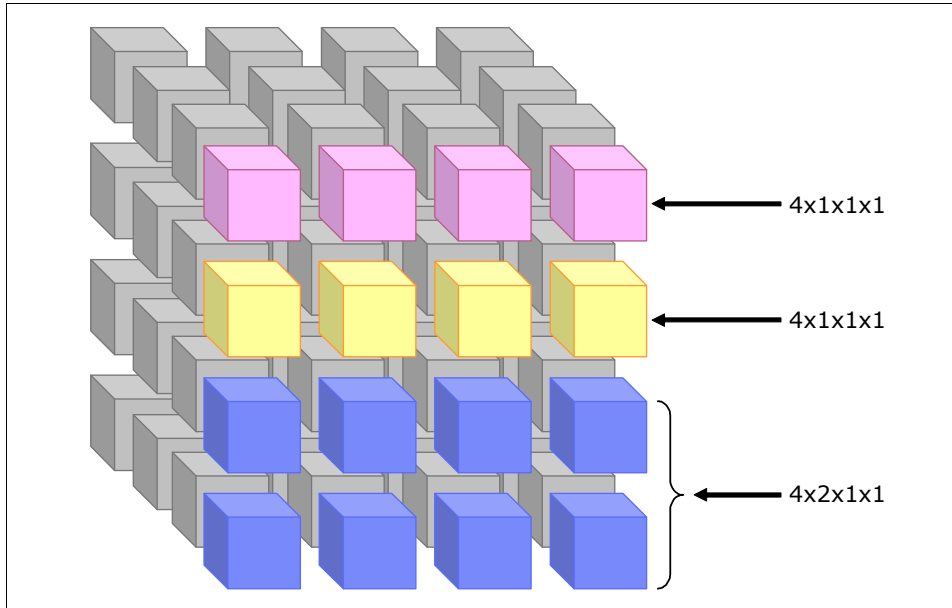


Figure 3-6 Two blocks each with four midplanes and one 8-midplane block

For any dimension, a torus can be formed by using all of the midplanes in that line or by wrapping the links and using only one complete midplane in that line. Additionally, using more than one midplane, but less than the entire line, means that the remaining midplanes will set their link chips to wrap/pass through for that dimension, and can achieve only a torus of size one. The top, front row of Figure 3-7 shows one 2x1x1 and two 1x1x1 blocks.

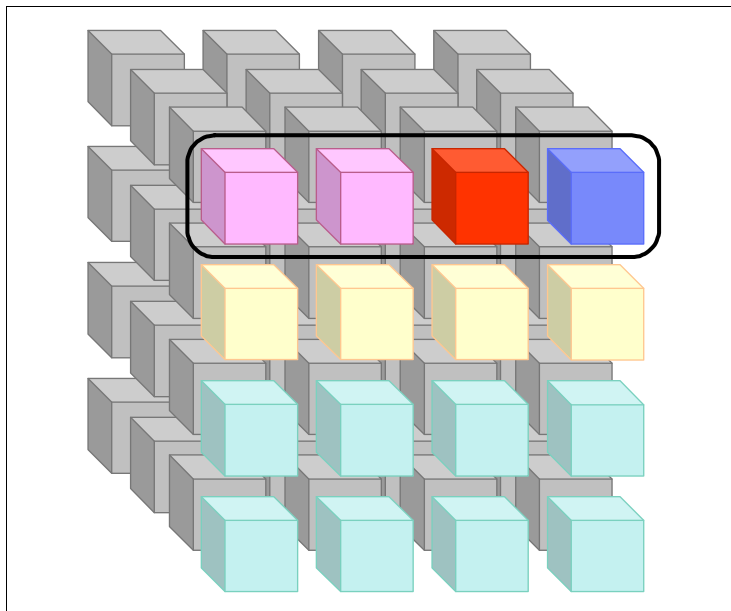


Figure 3-7 Two single midplane blocks and a single two midplane block

3.1.2 Small blocks

Small blocks are made up of one or more node boards within a single midplane. Small blocks are always a multiple of 32 nodes and cannot be a torus in all five dimensions. Small blocks

can be created in several ways: on the Blocks page in Navigator, by job schedulers that use the Blue Gene/Q Scheduler APIs, or with the `gen_small_block` command in `bg_console`. Table 3-1 shows the dimensions and connectivity for each of the supported small block configurations.

Table 3-1 Small block specifications

Number of node boards	Number of nodes	Dimensions	Torus directions (ABCDE)
1	32	2x2x2x2x2	00001
2 (adjacent pairs)	64	2x2x4x2x2	00101
4 (quadrants)	128	2x2x4x4x2	00111
8 (halves)	256	4x2x4x4x2	10111

The block size of 512 is not shown in Table 3-1 because that is a complete midplane (and a torus in all dimensions). Therefore, it is considered a large block.

3.2 Creating compute blocks

Compute blocks can be created in several ways: on the Blocks page in Navigator, by job schedulers that use the Blue Gene/Q Scheduler APIs, or by using various MMCS block creation commands in `bg_console`. This section describes creating compute blocks from `bg_console`.

There are commands in `bg_console` to create all types of blocks. This section covers only the commands that relate to creating compute blocks. Chapter 12, “Midplane Management Control System server” on page 147 goes into more detail about the remainder of the supported commands. The supported set of commands for generating blocks are:

- ▶ `gen_block`
- ▶ `gen_blocks`
- ▶ `gen_full_block`
- ▶ `gen_midplane_block`
- ▶ `gen_small_block`

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

The `gen_block` command

The `gen_block` command generates a block that consists of one midplane. The command uses the following syntax:

```
genblock <blockid> <midplane>
```

The parameter definitions are as follows:

- `<blockid>` The name of the block to be created.
- `<midplane>` The midplane to create the block on. For example, R00-M0.

The `gen_blocks` command

The `gen_blocks` command generates a block for each midplane on a machine. The command uses the following syntax:

```
gen_blocks [blockidprefix]
```

The `blockidprefix` is optional. If omitted, each block name is the position of the midplane.

The `gen_full_block` command

The `gen_full_block` command creates a block for the entire machine. The syntax of the command is:

```
gen_full_block <blockid>
```

The `gen_midplane_block` command

The `gen_midplane_block` command creates a block for a set of midplanes. The syntax of the command is:

```
gen_midplane_block <blockid> <corner> <asize> <bsize> <csize> <dsiz> [aPT>
<bPT> <cPT> <dPT>]
```

The parameter definitions are:

<blockid>	The name of the block to be created.
<corner>	The location of the first midplane included in the block. The location specified is the 0,0,0,0 position of the midplanes in the block.
<(a, b, c, d)size>	The size of the block in terms of the number of midplanes in A, B, C, D dimensions.
<(a, b, c, d)PT>	aPT,bPT,cPT,dPT are optional arguments to specify the use of pass through and are expected to be a string of 1s (include) and 0s (pass through).

The following example creates a block named R00A4B2 that uses pass-through. This example corresponds to the blue block in Figure 3-6 on page 34. The size is 4 x 2 x 1 x 1, and the last two midplanes in the B dimension are passed through to complete the torus.

```
gen_midplane_block R00A4B2 R00-M0 4 2 1 1 1111 1100 1 1
```

The `gen_small_block` command

The `gen_small_block` command generates a submidplane block. The syntax of the command is:

```
gen_small_block <blockid> <midplane> <cnodes> <nodeboard>
```

The parameter definitions are:

<blockid>	The name of the block to be created.
<midplane>	The location of the midplane that will contain the block.
<cnodes>	The total number of compute nodes to include in the block. The number must be 32, 64, 128, or 256.
<nodeboard>	The location of the node board that the compute nodes reside on. For a block size of greater than 32, this parameter is the starting node board for the multiple node board block.

3.3 I/O blocks

I/O nodes are used by the compute nodes to communicate with devices on networks external to the Blue Gene/Q racks. When compute blocks are booted, they must have a method to communicate, which is achieved by attaching to I/O blocks. When a compute block is booted, all of the linked compute nodes must be attached to a booted I/O node. Therefore, before booting a compute block, all connected I/O nodes must already be booted. The converse is also true. When freeing an I/O block, the attached compute blocks must be freed first.

For a complete description of I/O services see Chapter 4, “Configuring I/O nodes” on page 41. For more information about I/O hardware, see the *Blue Gene/Q Hardware Overview and Installation Planning Guide*, SG24-7822 Redbooks publication.

3.4 Creating I/O blocks

I/O blocks can be created on Blue Gene/Q using the `gen_io_block` command in `bg_console`. This section describes using that command to create an I/O block.

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

The `gen_io_block` command

The `gen_io_block` command generates a block for a set of I/O nodes. The syntax of the command is:

```
gen_io_block <blockid> <location> <ionodes>
```

The parameter definitions are:

<code><blockid></code>	The name of the block to be created.
<code><location></code>	Specify an I/O node, for example: Qxx-Ix-Jxx Rxx-Ix-Jxx Or an I/O drawer: Qxx-Ix Rxx-Ix
<code><ionodes></code>	The number of nodes to include in the block. The number must be a multiple of 8 when specifying an I/O drawer. The number of nodes must be 1, 2, or 4 when specifying an I/O node.

For two nodes, the node location must be J00, J02, J04, or J06. For four nodes, the node location must be J00 or J04.

The following example creates an I/O block named `IOBlock` with 128 I/O nodes included in the block. The 128 I/O nodes to include in the I/O block are determined by the starting I/O drawer location (`R00-ID` in this example). The remaining I/O nodes in the block come from the other I/O drawers in the machine, sorted alphabetically by their location string, until the block I/O node size is reached. Each I/O drawer contains eight I/O nodes, so the block will include I/O nodes from a total of 16 I/O drawers to meet the requested 128 I/O nodes:

```
gen_io_block IOBlock R00-ID 128
```

3.5 I/O links and ratios

Every compute node in a compute block is serviced by an I/O node. By default, the maximum supported ratio is 256 compute nodes to one (256:1) I/O node. This maximum is controlled by the *max_connected_nodes* key in the [cios] section of the bg.properties file. Any attempt to boot compute blocks over this limit will fail, as shown in Example 3-1.

Example 3-1 Exceeding I/O ratio

```
mmcs$ allocate mytest
FAIL
failed to update I/O node usage for mytest, R00-IC-J00 using 64 over limit of 8
mmcs$
```

These ratios can be affected by I/O nodes or I/O links becoming unavailable due to RAS events. See Chapter 7, “Reliability, availability, and serviceability” on page 93 for a description of the Control Actions.

I/O links for a compute block can be queried from the console using the `list_io_links` command, as shown in Example 3-2.

Example 3-2 list_io_links command

```
mmcs$ list_io_links R00-M0-N00
OK
COMPUTE NODE    I/O NODE        STATUS  LINK STATUS
R00-M0-N00-J11  R00-ID-J00      A       A
R00-M0-N00-J06  R00-ID-J01      A       A
mmcs$
```

Both the I/O node status and I/O links status must be (A)vailable for the link to be trained when the compute block is booted.

3.6 Deleting blocks

Blocks can be removed from the system when no longer needed. There are several ways to remove blocks from the Blocks tab: in the Blue Gene Navigator administrative web application, by using the Blue Gene/Q Scheduler APIs, and by using `bg_console`.

The following example uses the `bg_console delete` command to remove block R00A4B2:

```
delete bgqblock R00A4B2
```

3.7 Block status transitions

When booting or freeing I/O or compute blocks on Blue Gene/Q, the blocks transition to several states. A description of each status follows:

Free	Indicates that the block is available for use.
Allocated	Indicates that the block resources have been reserved. Next, the block transitions to Free status if the block is terminating or the block transitions to booting if processing a block boot request.

- Booting** Indicates that the block resources are in the process of booting up. Next, the block transitions to Terminating if the block boot fails, or the block transitions to Initialized if the block boot is successful.
- Initialized** Indicates that the block resources are initialized and ready for use. For compute blocks, this means that jobs can be run on the compute nodes of the block. For I/O blocks, this means that the I/O nodes can process I/O requests from jobs running on compute nodes.
- Terminating** Indicates that the block resources are in the process of shutting down.

Figure 3-8 shows the possible block statuses and transition steps.

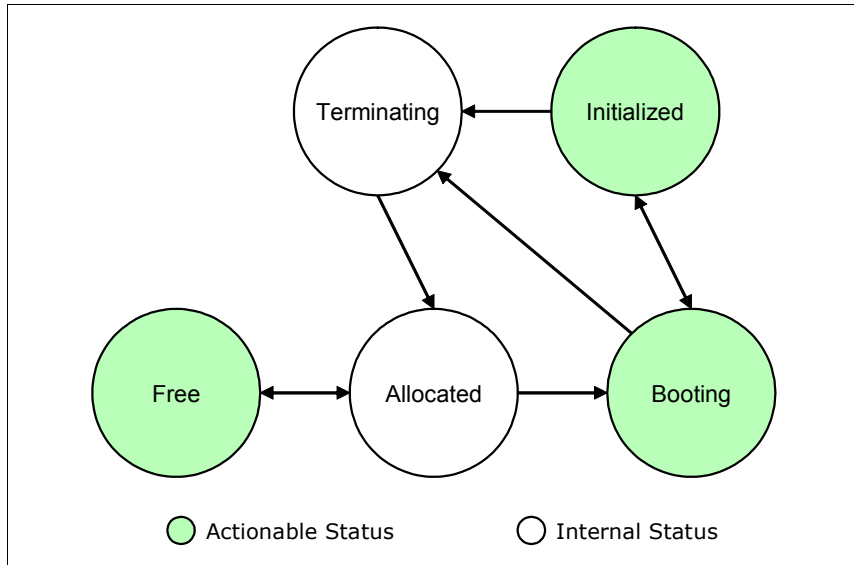


Figure 3-8 Block transition diagram



Configuring I/O nodes

This chapter contains information about:

- ▶ I/O node kernel:
 - Building new Linux kernels and installing Linux kernels from the Red Hat download web site
- ▶ I/O node run time:
 - Installing Red Hat Enterprise Linux (YACI, YUM, and so on) and additional packages
- ▶ I/O adapters:
 - Supported adapters
 - Firmware updates
- ▶ I/O configuration:
 - Defining interfaces and IP addresses
 - Environment variables through `/dev/bgpers`
 - Site customizations (mounting file systems)
 - Customizing ramdisk
- ▶ Boot sequence:
 - Persistent file systems
- ▶ Maintenance:
 - Node Health Monitor
- ▶ I/O services
- ▶ NFS configuration
- ▶ Troubleshooting:
 - I/O node log files, RAS, and so on

Note: Configuring GPFS in the Blue Gene/Q environment is beyond the scope of this book. For step-by-step instructions, see General Parallel File System HOWTO for the IBM System Blue Gene/Q Solution (SC23-6939-00) at:
<http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=SC23-6939-00>

4.1 I/O node kernel

The I/O node kernel is a patched Red Hat Enterprise Linux 6 kernel. The patches provide support for the Blue Gene/Q platform and contain a few modifications to improve performance. The precise version of the Red Hat Enterprise Linux kernel supported on Blue Gene/Q might change over time to keep current.

4.1.1 Building new Linux kernels

Although it is not necessary (or recommended) to rebuild the Linux kernel, it is possible to do so to integrate a critical security fix or add functionality that is not made available by Red Hat or IBM. In general, necessary kernel updates are provided by Red Hat or IBM. Unapproved Linux kernel modifications might not be supported.

4.1.2 Installing the Linux kernel from the Red Hat download website

Blue Gene/Q Linux kernel updates might be available from the Red Hat website. To download and install kernel updates, follow these steps:

1. From a browser, go to <https://access.redhat.com/home>
2. Hover over "Downloads" then select "Channels" from the menu that appears.
3. Log in when prompted.
4. On the leftmost text box under "Filter by Product Channel:" select "Red Hat Enhanced Kernel for Blue Gene" from the drop-down list.
5. On the architecture drop-down list (third from the left) choose "PPC".
6. Click the "Filter" button.
7. Click the + symbol next to "Red Hat Enterprise Linux Server 6" on the Channel Name list. Click the "PPC" Architecture link on the right side of the sub-channel "Red Hat Enhanced Kernel for Blue Gene Server 6".
8. Click the "Packages" tab.
9. In the "Filter by Package:" box enter a unique part of the desired kernel version string then click Go.
10. Click "Select All".
11. Click "Download Packages".
12. Click "Download Selected Packages Now!".
13. Save the `rhn-packages.tar` file to an appropriate location (for example, `/bgsys/downloads/rhn`).
14. Untar `rhn-packages.tar` to obtain the actual kernel RPMs (for example, `tar xvf rhn-packages.tar`).
15. Install the kernel RPMs into the I/O node distribution (for example, `/bgsys/drivers/ppcfloor/ramdisk/tools/bgqDistrofsPostInstall.sh -i -r /bgsys/downloads/rhn`).
16. Recompile all kernel modules that were previously installed in the distribution.

4.2 I/O node run time

Blue Gene/Q I/O nodes run a Red Hat Enterprise Linux 6 (RHEL6) based Linux distribution called the Blue Gene/Q Linux distribution. The Blue Gene/Q Linux distribution contains all of the same packages that exist on a PowerPC RHEL6 front end node plus some additional modules and programs to support running on Blue Gene hardware. The Blue Gene/Q Linux distribution is installed like a stand-alone Linux box into a directory under `/bgsys/linux`. This approach allows the Blue Gene/Q Linux distribution to track its installed software through RPM. The Blue Gene/Q Linux distribution installed under `/bgsys/linux` is used by all I/O nodes through NFS as a read-only root environment. The exact version of RHEL6 used for the Blue Gene/Q Linux distribution might vary based on the most recent release of RHEL6 supported on the Blue Gene/Q system.

It is possible to have multiple Blue Gene/Q Linux distribution images installed under `/bgsys/linux`. Each Blue Gene/Q Linux distribution is associated with a Blue Gene/Q driver. The latest Blue Gene/Q Linux distribution image is pointed to by `/bgsys/linux/ionfloor`.

4.2.1 Installing Red Hat Enterprise Linux

The Red Hat Enterprise Linux image that is used to create the Blue Gene/Q Linux distribution is installed automatically when the Blue Gene/Q software stack is installed on the service node. The following steps must first be executed to enable the automatic installation process.

YACI installation

The tool that actually installs the RHEL6 image onto the service node is called Yet Another Cluster Installer (YACI). YACI takes a list of RPMs and an installation source and creates a Linux distribution image. YACI does not come with RHEL6 by default, so users must install YACI on the service node before Blue Gene/Q software stack installation.

YACI is a free software package that can be downloaded from <http://www.yaci.org>. Download YACI version 12-7.4.ch4. After downloading YACI, you must install it using the command `rpm -ivh yaci-12-7.4ch4.noarch.rpm`.

RHEL6 repository setup

YACI requires a RHEL6 RPM repository to pull RPMs for installation. The repository should contain a copy of all the RPMs contained on the RHEL6 installation media. This repository is at `/install/redhat/RHEL6.2-ppc/Packages` or the location listed in the installation README instructions. Before installing any subsequent Blue Gene/Q software release, consult the installation README instructions because the newer RHEL6 repository might be required to be available under the `/install/redhat` directory. This repository location can either be local to the service node or mounted from a remote file server.

Check free file space on the service node

There must be sufficient file space on the service node to support installation of RHEL6 and subsequently the Blue Gene/Q Linux distribution. The root (`/`) file system on the service node must have at least 6 GB of free space. The `/bgsys` mount must have at least 12 GB of free space. If `/bgsys` is part of the root file system of the service node, a total of 18 GB of space must be available on root. The available free space on the service node's file systems can be found by using the `df -h` command. A pre-check occurs before installing RHEL6 to verify that enough space is available. The installation does not proceed if `/bgsys` does not have at least 12 GB of free space. This restriction prevents partial installations from occurring.

4.2.2 Installing additional packages

The Blue Gene/Q Linux distribution installation process automatically handles adding packages to the Blue Gene/Q Linux distribution. This processing happens as part of the execution of `/bgsys/drivers/<driver>/ramdisk/tools/bgqDistrofsPostInstall.sh`, which is executed during `bgq-distrofs` RPM installation. To enable this process, create an “addpkgs” directory under `/bgsys/linux`. The installation first attempts to install all the RPMs located in this directory. Then it attempts to execute all non-RPM executables in `/bgsys/linux/addpkgs`.

The `addpkgs` processing verifies that the RPMs are not installed in the Blue Gene/Q Linux distribution image for the current driver. Only RPMs that were not previously installed are installed. It is also possible to update previously installed RPMs. When updating to a newer RPM level, remove earlier RPMs from the `/bgsys/linux/addpkgs` directory to prevent the RPM tool from displaying warning messages about a newer package already being installed. The RPMs are installed using `–root` to install them into the appropriate Blue Gene/Q Linux distribution image. This results in the RPM files being seen in the shared NFS root space from each I/O node and also being seen in the shared root space RPM repository.

The executables found in `/bgsys/linux/addpkgs` are executed in alphabetical order. During this process, there are three environment variables that tell the executables which Blue Gene/Q Linux distribution image is being prepared.

DRIVER_PATH Specifies the driver installed under `/bgsys/drivers` that uses the Blue Gene/Q Linux distribution.

DRIVER_LINUX_OS_PATH Specifies the linux/OS link for the current driver (for example, `/bgsys/drivers/<driver>/ppc64/linux/OS`).

DISTROFS_LOC Specifies the Blue Gene/Q Linux distribution that is to be prepared (for example, `/bgsys/linux/RHEL6.2_V1R1M2-0`).

It can be useful to include scripts in the `addpkgs` directory. For example, the `localMounts.sh` script creates the required mount points in the Blue Gene/Q Linux distribution for additional user file systems that must be mounted to the I/O node. Scripting can also be included under `addpkgs` to automatically compile and install additional modules for the I/O node as is required for GPFS and other similar file systems.

It is possible to manually install RPMs into the Blue Gene/Q Linux distribution using `rpm –root`. However, this method is not recommended. Manually installing RPMs into the image changes only the selected Blue Gene/Q Linux distribution image. When subsequent images are created, those packages are not automatically included.

To force the reprocessing of the `addpkgs` directory against an existing Blue Gene/Q Linux distribution image, run the following command:

```
/bgsys/drivers/ppcfloor/ramdisk/tools/bgqDistrofsPostInstall.sh –a.
```

Using this option reprocesses only the RPMs and executables in `/bgsys/linux/addpkgs`.

4.2.3 Creating Blue Gene/Q Linux distribution sandboxes

A private Blue Gene/Q Linux distribution can be created for test purposes. To create a private sandbox, execute `/bgsys/drivers/ppcfloor/ramdisk/tools/bgqDistrofsPostInstall.sh –d /bgsys/linux/<Sandbox Directory>`. This causes a new Blue Gene/Q Linux distribution image to be created in the requested directory. The `/bgsys/linux/addpkgs` directory is automatically processed as part of the installation process. To boot the new Blue Gene/Q Linux distribution image, a new ramdisk must be built that points to the sandbox. A new ramdisk can be built with the following command:

```
/bgsys/drivers/ppcfloor/ramdisk/bin/build-ramdisk -o <ramdisk name> --runos
/bgsys/linux/<Sandbox Directory>.
```

Booting the ramdisk that is output by this command boots into the sandbox Blue Gene/Q Linux distribution image.

4.3 I/O adapters

InfiniBand and 10-gigabit Ethernet (10GbE) adapters are supported in Blue Gene/Q I/O drawers. The I/O drawers provide a PCI Express 2.0 x8 slot for adapter attachment.

4.3.1 Supported adapters

Three ConnectX-2 adapters from Mellanox are supported:

- ▶ A single port quad data rate (QDR) InfiniBand adapter (known as Kestrel)
- ▶ A dual port 10GbE adapter (known as Hawk)
- ▶ A dual port combination 10GbE/InfiniBand adapter (known as Raptor).

The supported adapter part numbers are shown in Table 4-1.

Table 4-1 Supported adapter part numbers

Adapter	Mellanox P/N	IBM P/N
Kestrel – Single Port IB	MHQH19B-XTR	74Y9057
Hawk – Dual Port 10GbE	MNPH29D-XTR	74Y9059
Raptor – Combo 10GbE/IB	MHZH29B-XTR	74Y9058

Other PCIe adapters installed in an I/O drawer cause a RAS event to be generated with each I/O node boot.

4.3.2 Firmware updates

Service actions against I/O drawers automatically ensure that the Mellanox PCIe adapters connected to that drawer have the latest Mellanox firmware applied.

Blue Gene/Q releases or efices provide the latest Mellanox firmware for the adapters supported on Blue Gene/Q. Check the Install README instructions to determine if a service action is required to apply new firmware after the installation of an efice or release upgrade.

You can view the latest available firmware levels at the following website:

http://www.mellanox.com/content/pages.php?pg=firmware_table_IBM

The firmware level on the installed adapters can be verified by checking the I/O node log files for the information which is printed at boot time. Example 4-1 shows example information.

Example 4-1 Example firmware information

```
{0}.10.0: Found supported PCIe adapter:
{0}.10.0:   Product Name: KESTREL QDR -
{0}.10.0:   [PN] Part number: MHQH19B-XTR - [EC] Engineering changes: A2 -
{0}.10.0:   FW Version: 2.9.1000 - PSID = MT_0D90110009.
```

4.4 I/O configuration

I/O nodes must have at least one external network adapter configured to properly boot. I/O nodes have one or more network interfaces configured depending on the system hardware configuration. An example network configuration on an I/O node is shown in Example 4-2.

Example 4-2 I/O node default network configuration

```
-bash-4.1# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN
    link/ether 0e:5c:be:b3:ff:0a brd ff:ff:ff:ff:ff:ff
3: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 00:02:c9:09:cb:4f brd ff:ff:ff:ff:ff:ff
4: ib0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 64000 qdisc pfifo_fast state UP qlen 250000
    link/infiniband 80:00:00:48:fe:80:00:00:00:00:00:00:00:02:c9:03:00:09:cb:4f brd
00:ff:ff:ff:ff:12:40:1b:ff:ff:00:00:00:00:00:00:ff:ff:ff:ff
    inet 172.20.212.20/16 brd 172.20.255.255 scope global ib0
    inet6 fe80::202:c903:9:ca3f/64 scope link
        valid_lft forever preferred_lft forever
```

Note: The external network interface is ib0 or eth0 and is on the subnet appropriate for the functional network on which the I/O nodes are connected.

4.4.1 Network Interface MTU

To maximize performance, set the network interface MTU to the largest size that your network can accommodate. For Ethernet networks, this size is typically 9000. For IB networks the ib0 interface is configured in connected mode to maximize performance. With connected mode, the MTU can be set as high as 65520. However, better TCP performance can be achieved when the MTU is 64000.

4.4.2 Defining interfaces and IP addresses

IP addresses are configured for I/O nodes by the IBM installation team. Normally no further action is required except for one situation. The single exception is in the case where the system is installed with dual port 10 GbE (Hawk) or combo 10 GbE/IB (Raptor) adapters. In this case, it might be desirable to add a network interface to the I/O nodes after the IBM installation team completes their work.

Blue Gene/Q provides a BGQNetConfig database table. This table contains the network configuration details for the I/O nodes in the system. Entries from this table are made available to nodes using the configuration service. The BGQNetConfig entries for a default I/O node are shown in Example 4-3.

Example 4-3 BGQNetConfig entries

```
[bgqadmin@bgqts6sn ~]$ /bgsys/drivers/ppcfloor/db/schema/dbNetConfig.pl --list --location
R00-IC-J03
Location      Interface      Name           Value
-----
```


R00-IC-J03	external1	ipv4address	172.20.212.20
R00-IC-J03	external1	ipv4broadcast	172.20.255.255
R00-IC-J03	external1	ipv4netmask	255.255.0.0
R00-IC-J03	external1	name	ib0

In Example 4-3, the I/O node has a Raptor combo adapter attached. To configure the Ethernet port on the I/O node, execute the commands shown in Example 4-4.

Example 4-4 Commands to configure Ethernet on I/O node

```

bgqadmin@bgqts6sn ~]$ /bgsys/drivers/ppcfloor/db/schema/dbNetConfig.pl --location
R00-IC-J03 --interface external2 --itemname ipv4address --itemvalue '172.16.212.20'
Inserted 1 rows into the BGQNetConfig
[bgqadmin@bgqts6sn ~]$ /bgsys/drivers/ppcfloor/db/schema/dbNetConfig.pl --location
R00-IC-J03 --interface external2 --itemname ipv4broadcast --itemvalue '172.16.255.255'
Inserted 1 rows into the BGQNetConfig
[bgqadmin@bgqts6sn ~]$ /bgsys/drivers/ppcfloor/db/schema/dbNetConfig.pl --location
R00-IC-J03 --interface external2 --itemname ipv4netmask --itemvalue '255.255.0.0'
Inserted 1 rows into the BGQNetConfig
[bgqadmin@bgqts6sn ~]$ /bgsys/drivers/ppcfloor/db/schema/dbNetConfig.pl --location
R00-IC-J03 --interface external2 --itemname name --itemvalue 'eth0'
Inserted 1 rows into the BGQNetConfig

```

After executing these commands, the `dbNetConfig.pl` script `--list` option for the same location now shows the configuration data for the second network interface, as shown in Example 4-5.

Example 4-5 BGQNetConfig entries with second network interface

```

[bgqadmin@bgqts6sn ~]$ /bgsys/drivers/ppcfloor/db/schema/dbNetConfig.pl --list --location
R00-IC-J03

```

Location	Interface	Name	Value
-----	-----	----	-----
R00-IC-J03	external1	ipv4address	172.20.212.20
R00-IC-J03	external1	ipv4broadcast	172.20.255.255
R00-IC-J03	external1	ipv4netmask	255.255.0.0
R00-IC-J03	external1	name	ib0
R00-IC-J03	external2	ipv4address	172.16.212.20
R00-IC-J03	external2	ipv4broadcast	172.16.255.255
R00-IC-J03	external2	ipv4netmask	255.255.0.0
R00-IC-J03	external2	name	eth0

After the node is rebooted, Example 4-6 shows that `eth0` and `ib0` network interfaces are now configured.

Example 4-6 I/O node eth0 and ib0 network configuration

```

-bash -4.1# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN
    link/ether 0e:5c:be:b3:ff:0a brd ff:ff:ff:ff:ff:ff
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP qlen 1000
    link/ether 00:02:c9:10:f4:80 brd ff:ff:ff:ff:ff:ff
    inet 172.16.212.20/16 brd 172.16.255.255 scope global eth0
    inet6 fe80::2:c900:109:ca3f/64 scope link
        valid_lft forever preferred_lft forever
4: ib0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 64000 qdisc pfifo_fast state UP qlen 250000

```

```

link/infiniband 80:00:00:48:fe:80:00:00:00:00:00:00:02:c9:03:00:09:cb:4f brd
00:ff:ff:ff:ff:12:40:1b:ff:ff:00:00:00:00:00:00:ff:ff:ff:ff
inet 172.20.212.20/16 brd 172.20.255.255 scope global ib0
inet6 fe80::202:c903:9:ca3f/64 scope link
valid_lft forever preferred_lft forever

```

For additional usage details for dbNetConfig.pl, see the dbNetConfig.pl man page, which can be accessed by executing `/bgsys/drivers/ppcfloor/db/schema/dbNetConfig.pl -h`.

4.4.3 Interface bonding configuration

Blue Gene/Q I/O nodes can be configured to use Ethernet interface bonding if the node has a Hawk dual port 10 Gb Ethernet adapter (part number MNPH29D-XTR or 74Y9059). To enable bonding, add kernel module options to the node configuration data. To add modules, use the dbNetConfig.pl script to add the BONDINGOPTS item name. For example, to configure bonding on the I/O node located at Q00-I2-J07 with the options 'mode=4 miimon=1000 xmit_hash_policy=layer3+4', execute `dbNetConfig.pl` as follows:

```

/bgsys/drivers/ppcfloor/db/schema/dbNetConfig.pl -location Q00-I2-J07 -interface
' -itemname BONDINGOPTS -itemvalue 'mode=4 miimon=1000 xmit_hash_policy=layer3+4'

```

This command causes the BONDINGOPTS variable to be added to the node's personality. The bond0 script, which can be found in `initramfs/etc/init.d`, finds BONDINGOPTS to be non-null. It then inserts the bonding.ko module using the options found in BONDINGOPTS. After the module is inserted, the external1 IP address from the BGQNetConfig table is used as the bond0 IP address.

4.4.4 Environment variables using /dev/bgpers

A Blue Gene/Q I/O node's personality is available through `/dev/bgpers`. This char device is a combination of the node's personality and configuration information. The first part of the output from `/dev/bgpers` for the node configured in section 4.4.2, "Defining interfaces and IP addresses" on page 46 is shown in Example 4-7.

Example 4-7 Output from /dev/bgpers

```

-bash-4.1# cat /dev/bgpers
BG_UCI=7800c0c000000000
BG_LOCATION=R00-IC-J03
BG_BLOCKID=1241
BG_GATEWAY=172.20.2.41
BG_IPV4_SN=172.20.2.1
BG_IPV6_SN=0:0:0:0:0:0:0:0
BG_BGSYS_IPV4=172.20.2.1
BG_BGSYS_IPV6=0:0:0:0:0:0:0:0
BG_BGSYS_EXPORT_DIR="/bgsys"
BG_BGSYS_EXPORT_MOUNT_OPTS=""
BG_DISTRO_IPV4=0.0.0.0
BG_DISTRO_IPV6=0:0:0:0:0:0:0:0
BG_DISTRO_EXPORT_DIR=""
BG_DISTRO_MOUNT_OPTS=""
BG_INTFO_NAME=ib0
BG_INTFO_MAC=00:00:00:00:00:00
BG_INTFO_IPV4=172.20.212.20
BG_INTFO_IPV6=0:0:0:0:0:0:0:0
BG_INTFO_NETMASK=255.255.0.0

```

```

BG_INTF0_IPV6_PREFIX=
BG_INTF0_BROADCAST=172.20.255.255
BG_INTF0_MTU=64000
BG_INTF1_NAME=eth0
BG_INTF1_MAC=00:00:00:00:00:00
BG_INTF1_IPV4=172.16.212.20
BG_INTF1_IPV6=0:0:0:0:0:0:0:0
BG_INTF1_NETMASK=255.255.0.0
BG_INTF1_IPV6_PREFIX=
BG_INTF1_BROADCAST=172.16.255.255
BG_INTF1_MTU=9000
...

```

Output from the personality is in an environment variable format so that `/dev/bgpers` can be sourced for use in bash scripts on the I/O node using the command `. /dev/bgpers`. The personality data can also be added to a script along with variables for the I/O node's RAS events and a number of other functions by sourcing `/etc/init.d/functions`.

It is possible to add environmental variables to the I/O node's personality. For example, to add the environment variable `SITE_FQDN` to provide the last part of the fully qualified domain name for use by site network configuration scripts on the I/O node. This additional environment variable is set by using the `bgNetConfig.pl` script, as shown in Example 4-8.

Example 4-8 Adding environment variable to /dev/bgpers

```

[bgqadmin@bgqts6sn bin]$ /bgsys/drivers/ppcfloor/db/schema/dbNetConfig.pl --location
R00-IC-J03 --interface "" --itemname SITE_FQDN --itemvalue 'rchland.ibm.com'
Inserted 1 rows into the BGQNetConfig
[bgqadmin@bgqts6sn bin]$ /bgsys/drivers/ppcfloor/db/schema/dbNetConfig.pl --list --location
R00-IC-J03

```

Location	Interface	Name	Value
-----	-----	----	----
R00-IC-J03		SITE_FQDN	rchland.ibm.com
R00-IC-J03	external1	ipv4address	172.20.212.20
R00-IC-J03	external1	ipv4broadcast	172.20.255.255
R00-IC-J03	external1	ipv4netmask	255.255.0.0
R00-IC-J03	external1	name	ib0
R00-IC-J03	external2	ipv4address	172.16.212.20
R00-IC-J03	external2	ipv4broadcast	172.16.255.255
R00-IC-J03	external2	ipv4netmask	255.255.0.0
R00-IC-J03	external2	name	eth0

Enclose itemvalues that include spaces in single (') or double (") quotes. If an itemvalue has spaces and no enclosing quotes, single quotes are automatically added to prevent errors when sourcing the personality.

User environment variables must not be associated with an interface and the itemname is limited to 16 characters. After the next boot of the I/O node, the new environment variable is appended to the end of the `bgpers` output, as shown in Example 4-9.

Example 4-9 Output from /dev/bgpers with added environment variable

```

-bash-4.1# cat /dev/bgpers
...
BG_NODE_ENABLE_ENVMON=1
BG_NODE_ENABLE_PCIE=1
BG_NODE_ENABLE_DIAGNOSTICSMODE=0
BG_DDR_SIZE_MB=4096

```

```
BG_CLOCK_MHZ=1600
SITE_FQDN=rchland.ibm.com
```

The user environment variable can be deleted using dbNetConfig.pl and the -clear option instead of the -itemvalue option.

4.4.5 Site customization

Site customization of I/O nodes happens through /bgsys/linux/bgfs. The files in this directory are processed at boot time and are used to make per boot customizations of the I/O nodes.

A common use for /bgsys/linux/bgfs is to add a script that mounts any file systems that are site unique. This can be accomplished by creating a sitefs script with the appropriate commands to mount those file systems in /bgsys/linux/bgfs/etc/rc.d/init.d. Then link /bgsys/linux/bgfs/etc/rc.d/rc3.d/S27sitefs to that script. You must add a script to /bgsys/linux/addpkgs to create any custom mount points within the Blue Gene/Q Linux distribution because the root of the Blue Gene/Q Linux distribution is mounted read-only at the time the sitefs script is executed during boot.

4.4.6 Customizing ramdisk

Blue Gene/Q uses the build-ramdisk utility in /bgsys/drivers/ppcfloor/ramdisk/bin to build custom ramdisks. A custom root source for the ramdisk can be specified with the -rootfs option. If no rootfs is specified, the default tree at /bgsys/drivers/ppcfloor/ramdisk/initramfs is used. The most common reason for building a custom ramdisk is to specify a different Blue Gene/Q Linux distribution image using the -runos image or to add additional files to the ramdisk using the -addtree option. The -addtree option allows you to add files/directories to the root of the ramdisk and symbolic links within the ramdisk. When a file added using addtree is the same as one being included from the rootfs, the addtree file is the one pulled into the ramdisk. Example 4-10 shows how IBM uses the build-ramdisk utility with -addtree to build the ramdisk in the driver to update firmware.

Example 4-10 Building customized ramdisk

```
[root@bgqts3sn bin]# ./build-ramdisk --addtree /tmp/firmwareUpdate/ -o
/tmp/mlxFirmwareUpdate--runos/bgsys/linux/RHEL6.2_sandbox--verbose
build-ramdisk: root file system:
/bgsys/drivers/DRV2011_1215_2330/ppc64/ramdisk/initramfs
build-ramdisk: runos: /bgsys/linux/RHEL6.2_sandbox
build-ramdisk: init:
build-ramdisk: new files/directories from /tmp/firmwareUpdate:
  bin/
  /sbin/
  /usr/
  /usr/bin/
  /lib64/
  /etc/
  /etc/rc.d/
  /etc/rc.d/rc3.d/
  /etc/rc.d/init.d/
  /etc/rc.d/rc3.d/S02firmwareUpdate -> ../init.d/firmwareUpdate
  /bin/fw-25408-2_9_1000-46M2203.bin
  /sbin/bghalt
  /usr/bin/mstflint
  /lib64/libm.so.6
```

```

/lib64/libz.so.1
/etc/rc.d/init.d/firmwareUpdate
build-ramdisk: final dirs in ramdisk:
  /bgusr
  /bin
  /dev
  /dev/pts
  /dev/shm
  /etc
  /etc/rc.d
  /etc/rc.d/init.d
  /etc/rc.d/rc3.d
  /lib
  /lib64
  /proc
  /sbin
  /selinux
  /sys
  /tmp
  /usr
  /usr/bin
  /usr/lib64
  /var
  /var/empty
  /var/empty/sshd
  /var/lib
  /var/lib/nfs
  /var/lib/nfs/rpc_pipefs
build-ramdisk: writing /tmp/ramdisk.gencpio.script.29388 for gen_init_cpio
build-ramdisk: compress /tmp/mlxFirmwareUpdate.cpio
build-ramdisk: build /tmp/mlxFirmwareUpdate from /tmp/mlxFirmwareUpdate.cpio.gz

```

Notice the list of files at the beginning of the output. It shows the files in the addtree, including a symbolic link and indicating that a new runos of /bgsys/linux/RHEL6.2_sandbox is being used. The output ramdisk, /tmp/mlxFirmwareUpdate contains those additional files and uses the sandbox Blue Gene/Q Linux distribution as the NFS root image.

4.4.7 The bgras command

The **bgras** command generates RAS events from the I/O node and is located in the /bin directory.

You can use the **cat /dev/bgras** command to display a list of valid RAS IDs for Linux on Blue Gene/Q, as shown in Example 4-11.

Example 4-11 Displaying RAS IDs

```

-bash-4.1# cat /dev/bgras
BGRAS_ID_PCIE_UNSUPPORTED_ADAPTER=0xa0001
BGRAS_ID_PCIE_MISSING_ADAPTER_VPD=0xa0002
BGRAS_ID_BGSYS_MOUNT_FAILURE=0xa0003
BGRAS_ID_GPFS_START_FAILURE=0xa0004
BGRAS_ID_NO_NETWORK_INTERFACE_DEFINED=0xa0005
BGRAS_ID_SCRIPT_FAILURE=0xa0006
BGRAS_ID_BGQ_DISTRO_MISSING=0xa0007
BGRAS_ID_GPFS_INIT_FAILURE=0xa0008
BGRAS_ID_PCIE_LINK_DEGRADED=0xa0009
BGRAS_ID_NETWORK_CONFIG_FAILURE=0xa000a

```

```

BGRAS_ID_INT_VECTOR_FAILURE=0xa000b
BGRAS_ID_GPFS_HOSTNAME_FAILURE=0xa000c
BGRAS_ID_KERNEL_PANIC=0xa000d
BGRAS_ID_ETHERNET_LINK_TIMEOUT=0xa000e
BGRAS_ID_IB_LINK_TIMEOUT=0xa000f
BGRAS_ID_NODE_HEALTH_MONITOR_WARNING=0xa0010
BGRAS_ID_ETHERNET_LINK_LOST=0xa0011
BGRAS_ID_IB_LINK_LOST=0xa0012
BGRAS_ID_ROOT_FS_UNRESPONSIVE=0xa0013

```

Like the `bgpers` device, the output of `/dev/bgras` is in a bash environment variable format, which makes it possible to source `/dev/bgras` or `/etc/init.d/functions` and use the environment variable names to generate RAS events instead of the hex RAS IDs. The most common RAS event for user scripts to use is the script failure RAS event. For example, a failure in the `sitefs` script can generate a RAS event, as shown in Example 4-12.

Example 4-12 Generating a RAS event

```

-bash-4.1# . /etc/init.d/functions
-bash-4.1# bgras $BGRAS_ID_SCRIPT_FAILURE "Failure mounting sitefs file system /scratch"

```

The command in Example 4-12 on page 52 causes a RAS event to be added to the database. The RAS event looks similar to the following example:

```

ID:          8610407
Time:        2011-12-20 16:33:42.792916
Message ID:  000A0006
Category:    Software_Error
Component:   LINUX
Severity:    WARN
Block ID:    R00-IC-J02
Location:    R00-IC-J02
ECID:        01852400601A070E080CC535372000A3000000000162800000000000000000000
Serial Number: 74Y8125YL3BK031400A
CPU:         1
Message:     [LINUX] An init script encountered an error or failed to properly execute: Failure
            mounting sitefs file system /scratch
Description: The init process was not able to fully execute one of the init scripts.
Service Action: Consult the node logs for further information regarding this failure and correct the
            indicated problem.
Raw Data:    BG_MBOX_TYPE=ASCII; BG_MBX=7800c08000000000 000a0006 Failure mounting
            sitefs file system /scratch;

```

As shown in Example 4-12 on page 52, the `bgras` command takes one or two arguments. The first argument is the message ID for the desired RAS event. If `/dev/bgras` was sourced, the ID environment variable can be used as shown in Example 4-12 on page 52. The second argument is the text string to be displayed in the RAS message. If no text string is provided on the command line, it is expected that the RAS message is provided through stdin. If `/etc/init.d/functions` was sourced by a script, using the `bgras` command also causes the RAS message to be echoed to the I/O node console and to the node log file.

4.5 Boot sequence

The boot process starts with a ramdisk that does the critical initial configuration of the node. The ramdisk is only briefly used before execution transfers to a read-only, shared, NFS root image. Most of the boot process executes from the NFS root image.

Site customization of I/O nodes is handled by the `etc/init.d/popbgfs` script, which is run twice during the boot process. The first time it runs from within the ramdisk. The script pulls forward files from `/bgsys/linux/bgfs/etc` to ensure that they are integrated early enough to be executed as part of the boot after execution transfers to the read-only NFS root space. The second time `popbgfs` executes is after the boot moves to the read-only NFS root. At this point, the script must complete the integration of `/bgsys/linux/bgfs` into the running system.

The `popbgfs` script processes all of the files in `/bgsys/linux/bgfs`. If a `var` or `root` directory is found in `/bgsys/linux/bgfs`, the script copies the files found there into the read/write tmpfs spaces created on each node for those directories. For any other directories found under `/bgsys/linux/bgfs`, their contents are copied in `/bgfs` on the I/O node. For example, if there is a `/bgsys/linux/bgfs/bin` directory containing some binaries, that directory and binaries are available on the I/O node at `/bgfs/bin`. In addition to copying the directories from `/bgsys/linux/bgfs` to the I/O node, the `popbgfs` script also extracts any `tar.gz` files into `/bgfs` on the I/O node. This makes it possible to add shared libraries, such as the Python shared libraries, into the I/O node's memory for quick access. A similar process is executed for the Blue Gene/Q Toolchain libraries and the Comm libraries so that the shared libraries can be quickly accessed in the I/O node's memory.

While booting in the ramdisk, basic information about the I/O node being booted is printed. The `external1` and `external2` (if defined) network interfaces are configured, `bgsys` is mounted, and the first stage of `popbgfs` is executed. Any failures encountered in configuring the `external1` network device, mounting `bgsys`, or finding the Blue Gene/Q Linux distribution cause a RAS event to be generated and appropriate debug data to be printed. To facilitate further debug, a user can force the ramdisk to not transfer the boot to NFS root by setting `RAMDISK_DEBUG=1` in the `BGQNetConfig` table for the nodes in question. When the "Starting udev:" message is displayed, the node successfully transferred to the shared NFS root space.

Init continues to process the combined result of the init scripts from the Blue Gene/Q Linux distribution and the additional start scripts pulled from `/bgsys/linux/bgfs`. Output from the init scripts is printed to the console and to the node's log file using the standard RHEL6 format. At the end of the boot process, the Control System is signaled that the boot successfully completed. The welcome message is printed, and the console is automatically logged in as root, as shown in Example 4-13.

Example 4-13 Boot process messages

```
Sending Boot Completion Signal to the Control System: [ OK ]
Red Hat Enterprise Linux Server release 6.2 (Santiago)
Kernel 2.6.32-220.el6.bgq110_20120220.ppc64 on an ppc64
bgqts6sn-R00-IC-J03 login: root (automatic login)
```

4.5.1 Persistent file systems

Blue Gene/Q provides a persistent file space for each I/O node. The persistent file systems are created under `/bgsys/linux/nodefs`, if they do not exist, by the `/etc/init.d/nodefs` start script in the Blue Gene/Q Linux distribution. The first time the node is booted the directories are initially populated with the contents of the same directory in the Blue Gene/Q Linux distribution. On subsequent boots, the persistent directory is bind mounted to its counterpart in the Blue Gene/Q Linux distribution. The list of directories that are persistent can be modified by changing the list of directories that are being acted upon in the `nodefs` start script.

Under `/bgsys/linux/nodefs`, there is a directory for each I/O node IP address. Under that directory, exists the persistent directories for that node. When an I/O node has both an IPv4 and IPv6 address configured, the IPv4 address will be used for the I/O node's persistent file

space. The `var/mmfs` directory is persistent for systems using GPFS and the `var/lib/random-seed` directory is persistent for each I/O node to help improve each node's entropy. The following files are also bind mounted to the persistent file space: `var/log/messages`, `var/log/btmp`, `secure`, and `wtmp`. In addition, the `var/spool/abrt` directory is bind mounted to the persistent file space to keep core files out of the I/O node memory.

4.6 Maintenance

The same procedures used to maintain the RHEL6 front end nodes and service nodes are used to maintain the Blue Gene/Q Linux distribution images. Any subsequent release of the Blue Gene/Q software follows the latest RHEL6 where possible.

4.6.1 Installing RHEL updates

To install a RHEL update, you must configure the `/etc/yum.repos.d/RHEL_6.2.repo` file under the desired Blue Gene/Q Linux distribution to point to the RHEL repository. After the repository is configured, the distribution can be updated by chrooting into the Blue Gene/Q Linux distribution and running `yum update`. Some of the scripts run during the update process might return warnings or errors because they are not executing in a running system. These warnings and errors do not negatively affect the state of the system.

4.6.2 Installing a new RHEL dot release

Blue Gene/Q follows recent releases of RHEL when possible, which means that subsequent releases of the Blue Gene/Q software stack are likely to require the availability of a new RHEL package repository to properly install. It is important to review the *Install README* and *Memo To Users* documents that ship with new releases of the Blue Gene/Q software stack. Ensure that the following requirements are satisfied before installing a new release:

- ▶ Ensure that the `/tftpboot` directory that YACI uses is still located somewhere that has sufficient space to create a full RHEL6.x distribution image (~6 GB).
- ▶ Verify that `/bgsys` has approximately 12 GB of space available. A pre-check is done before installing RHEL6 to verify that enough space is available. The installation does not proceed if `/bgsys` does not have at least 12 GB of free space. This restriction prevents partial installations from occurring.
- ▶ Ensure that any additional packages in `/bgsys/linux/addpkgs` are compatible with RHEL6.x. If not, update these packages before installing the RHEL6.x based driver.
- ▶ `/install/redhat/RHEL6.x-ppc/Packages` must have a copy of all the RHEL6.x RPMs.
- ▶ When a new driver is installed, be aware that the process takes a significant amount of time to install the `bgq_distrofs` RPM. The `bgqDistrofsInstall` log file for RHEL6.x_base under `/bgsys/logs/BGQ` tracks the progress of the installation.

4.6.3 Installing efixes

Any efix that includes the `bgq-distrofs` RPM will cause a new Blue Gene/Q Linux distribution to be built under `/bgsys/linux`. The new Blue Gene/Q Linux distribution will be named based on the efix that installs it. For example, if `V1R1M2 eFix002` included `bgq-distrofs`, the RPM installation process would build a new Blue Gene/Q Linux distribution in the `/bgsys/linux/RHEL6.2_V1R1M2-2` directory and apply the changes from the efix to that

image. The directory `/bgsys/drivers/ppcfloor/linux/OS` is also updated to point to the new Linux image as part of the efix installation process.

The Control System must be stopped during the installation of any efix that includes the `bgq-distrofs` RPM because the booting of I/O blocks during the process can cause unpredictable installation results. To ensure that the changes are automatically applied as part of the efix installation process, make all site customizations with scripts or packages in `/bgsys/linux/addpkgs`.

When installing an efix using `rpm -Uvh` or `yum localinstall` the final step of the process is to uninstall the previous `bgq-distrofs` RPM. This means that the existing Blue Gene/Q Linux distribution will be removed at the end of the efix installation. For example, if the system currently has the `bgq-distrofs-1.1.2-0.ppc64.rpm` installed and `eFix002` comes with the `bgq-distrofs-1.1.2-2.ppc64.rpm`, this will cause `/bgsys/linux/RHEL6.2_V1R1M2-2` to be installed and `/bgsys/linux/RHEL6.2_V1R1M2-0` to be removed. Before installing an efix, it is important to ensure that `/bgsys/linux/ionfloor` is pointing to the same Blue Gene/Q Linux distribution pointed to by `/bgsys/drivers/ppcfloor/linux/OS`. If an administrator unintentionally used `bgqDistrofsInstall.sh -f` since the last release or efix was installed, or if a development driver has been loaded on the system, the `ionfloor` link might be incorrect. The `ionfloor` link is used by the efix installation process to determine the previously installed Blue Gene/Q Linux distribution. If the link is not properly set before efix installation this can result in an older distribution being incorrectly left on the system or can also result in the efix removing its own Blue Gene/Q Linux distribution.

4.6.4 Installing GPFS updates

Updates to GPFS for I/O nodes follow Blue Gene/Q software releases when possible. It is important to review the *Install README* and *Memo To Users* documents to determine if a new level of GPFS should be installed with a Blue Gene/Q software update. Note that updating GPFS on service nodes and front end nodes is outside the scope of this Redbooks publication. It is assumed those updates are handled before completing this process. If new GPFS RPMs are available, they should be downloaded and placed in `/bgsys/linux/addpkgs`, replacing the previous GPFS RPMs, before the installation of new Blue Gene/Q RPMs. This process ensures that the new GPFS RPMs can be installed as part of the Blue Gene/Q RPM installation.

If updated GPFS RPMs are released independently from a Blue Gene/Q software release, place the new RPMs in `/bgsys/linux/addpkgs`, replacing the previous GPFS RPMs, and install them with the following command:

```
/bgsys/drivers/ppcfloor/ramdisk/tools/bgqDistrofsPostInstall.sh -a.
```

If previously installed GPFS RPMs have been removed from the Blue Gene/Q Linux image (using the command `rpm --root /bgsys/linux/RHEL6.2_V1R1M2-0 -e <GPFS RPMs>`) the GPFS RPMs must be reinstalled. This is done by placing the updated RPMs in `/bgsys/linux/addpkgs` as a replacement to the previous RPMs and running the command `/bgsys/drivers/ppcfloor/ramdisk/tools/bgqDistrofsPostInstall.sh -i` to install the new RPMs.

For each scenario described above, it might still be necessary to compile GPFS in the Blue Gene/Q Linux distribution after installation has completed.

If GPFS fails to start after an update, perform the following tasks:

- ▶ Verify the service node is able to mount GPFS file systems. If the service node also fails to mount, solve the problem on the service node first.
- ▶ Verify that the `GPFS_STARTUP=1` flag is set in `/bgsys/linux/bgfs/etc/sysconfig/gpfs`.

- ▶ Verify that the gpfs init script exists in the `/bgsys/drivers/ppcfloor/linux/OS/etc/init.d/` directory. If the init script is not found then restore the script by executing `/bgsys/drivers/ppcfloor/ramdisk/tools/bggDistrofsPostInstall.sh -i`. If the script exists, but is a link to `/usr/lpp/mmfs/bin/gpfsrunlevel`, this means that GPFS was not compiled for the correct architecture. Correct this by removing the link and then running `/bgsys/drivers/ppcfloor/ramdisk/tools/bggDistrofsPostInstall.sh -i`. Ensure that GPFS is recompiled from `/bgsys/drivers/ppcfloor/linux/OS/usr/lpp/mmfs/src` after `bggDistrofsPostInstall.sh` completes the installation .
- ▶ If the previous steps do not resolve the problem, consult the GPFS log files under the `/bgsys/linux/nodefs/<l/O node IP address>/var/adm/ras` directory.

4.6.5 Node Health Monitor

The Blue Gene/Q system provides an active health monitor to the Linux nodes running on the system. The role of the Node Health Monitor is to monitor the state of vital system resources on the Linux nodes. If any of the resources are outside the configured thresholds, the Node Health Monitor produces a RAS event describing the problem condition and dumps a summary of the state of the node. The Node Health Monitor runs from a chrooted shell in the node memory allowing it to continue to run even after the node becomes unresponsive due to networking or NFS root problems. The thresholds and functionality of the Node Health Monitor are fully configurable. To modify the thresholds, copy the `bghealthmond` configuration file in `/bgsys/drivers/ppcfloor/ramdisk/initramfs/etc/sysconfig` to `/bgsys/linux/bgfs/etc/sysconfig` and make the required updates. The following system attributes are monitored:

- ▶ Free memory (MEMTHRESHOLD) – Default threshold: 256000 kb
- ▶ Transmit and receive errors on all network adapters (NETERRTHRESHOLD) – Default threshold: 25 packets
- ▶ Dropped transmit and received packets on all network adapters (NETDRPTHRESHOLD) – Default threshold: 2% of the total transmitted or received packets.
- ▶ Five-minute load average (LOADAVGTHRESHOLD) – Default threshold: 500
- ▶ Open file descriptors (OFTHRESHOLD) – Default threshold: 90% of the total possible open files.
- ▶ NFS retransmissions (RETRANSTHRESHOLD) – Default threshold: 30 packets
- ▶ InfiniBand and/or Ethernet link status (LINKRETRIES and LINKRETRYSLLEEP) – Default threshold: Link down for two minutes (four retries with 30 second sleeps in between).
- ▶ Readability of the NFS root file system (ROOTFSCHECKSLEEP) – Default threshold: An attempt to read from the file system taking more than 30 seconds to complete.
- ▶ A check for duplicate IP addresses is executed for the external adapters configured upon the node.

In addition to the configurable thresholds above, the following additional configuration items might also be configured:

- ▶ **ENABLED:** This item is set to '1' by default. If set to '0', the Node Health Monitor is not started on nodes.
- ▶ **FREQUENCY:** Configures the number of seconds to sleep between iterations of the Node Health Monitor. This item is set to 300 by default.
- ▶ **VERBOSE:** This item is set to '0' by default. If set to '1' the Node Health Monitor prints its execution progress with each iteration.

- ▶ LOGPATH: By default, the logpath is set to /dev/console causing any Node Health Monitor events to be logged to the console and then to the node log file in /bgsys/logs/BGQ. This path can be changed to another writeable space accessible by the node, which causes a file named bghealthmond-event-`<date>.log` to be created under the requested directory for each Node Health Monitor event. The logfile created is then indicated in the RAS event for future reference.

When any of the configured thresholds are exceeded, a Node Health Monitor RAS event (ID 0xA0010) is generated. An event is logged to the console or selected log file. The event includes:

- ▶ The output of /proc/meminfo
- ▶ The output of **top**
- ▶ The current network status for all the adapters
- ▶ The number of processes running
- ▶ The output of **nfsstat**
- ▶ If the problem encountered is not an NFS failure, the number of open file descriptors

Note: Network errors (dropped and errored packets) are only logged once to avoid spamming the RAS database and log files with such events. In addition, to reduce the total RAS output, the following processes from **top** are removed: ksoftirqd, migration, watchdog, events, kintegrityd, kblockd, md, rpciod, aio, crypto, infiniband, ib_cm, kthrotld, md_misc, and grep.

4.6.6 Core files

Blue Gene/Q I/O nodes have core file generation enabled by default. Core files, as on service nodes and front end nodes, are dumped to /var/spool/abrt using core files named core.<executable>.<pid>. I/O nodes do not use abrt. Instead, the abrt directory has the sticky bit set to provide accessibility for all users while maintaining a level of security. The /var/spool/abrt directory is part of the node's persistent file space so that the core files can be accessed through /bgsys/linux/nodefs/<node ip>/var/spool/abrt regardless of the state of the node. This prevents the node's memory from being exhausted by programs dumping many core files on the I/O node. For I/O nodes, /proc/sys/fs/suid_dumpable is set to '1' and ulimit -c set is set to 'unlimited'. These settings can be overridden by putting custom /etc/sysctl.conf and /etc/security/limits.conf files in /bgsys/linux/bgfs.

4.7 Common I/O services

Common I/O services (CIOS) provide the I/O subsystem for a Blue Gene/Q system. The Compute Node Kernel (CNK) uses CIOS for job control, function shipped I/O operations (including standard I/O), and running tools. CIOS includes the following services:

- ▶ I/O services daemon (IOSD) is the service that manages and monitors the other services. It is started when the I/O node is booted. There is one IOSD per I/O node. IOSD is configurable to start daemons for other services.
- ▶ Job control daemon (JOBCTLD) is the service used by both CNK and runjob_server for job control. It is started when the I/O node is booted. There is one JOBCTLD per I/O node. JOBCTLD is mostly a bridge between CNK and runjob_server for loading, starting, signaling, and ending jobs.
- ▶ Standard I/O daemon (STDIOD) is the service used by both CNK and runjob_server for standard I/O. It is started when the I/O node is booted. There is one STDIOD per I/O node.

STDIOD is mostly a bridge between CNK and runjob_server to receive standard input data and send standard output and standard error data.

- ▶ System I/O daemon (SYSIOD) is the service used by CNK to provide default function shipped I/O. It is started when the compute node is booted. There is one SYSIOD for each physical compute node, and it is shared by all of the processes running on a compute node. SYSIOD is a thin layer that simply invokes system calls on behalf of the compute node process.
- ▶ Tool control daemon (TOOLCTLD) is the service used by CNK to coordinate access to compute node processes among tools. It is started when the compute node is booted. There is one TOOLCTLD for each physical compute node and it is shared by all of the processes running on a compute node. Tools provide additional services to CNK. A debugger is the primary example of a tool.

4.7.1 Configuration

The CIOS daemons can be configured by setting options in the Blue Gene configuration file or by setting command-line options when the server starts. There are no configuration options for the tool control daemon.

CIOS configuration file options

Configuration options used by all CIOS daemons are in the [cios] section of the Blue Gene configuration file (/bgsys/local/etc/bg.properties). Configurable options include:

large_region_size = 1048576

Size in bytes of large memory regions used for I/O operations. Valid values are 65536 (64 KiB) to 8388608 (8 MiB). The value should be a power of 2. If large_region_size is invalid or not specified, the default value is used (1048576).

IOSD configuration file options

Configuration options used by the I/O services daemon are in the [cios.iosd] section of the Blue Gene configuration file (/bgsys/local/etc/bg.properties). Configurable options include:

listen_port = 7001

The port number used to listen for a Control System connection. If listen_port is invalid or not specified, the default value is used (7001).

jobctl_daemon_path = /sbin/jobctld

Path to job control daemon executable. There is one job control daemon per I/O node and it is started by IOSD when the I/O node is booted. If jobctl_daemon_path is not specified, the default value is used (/sbin/jobctld).

stdio_daemon_path = /sbin/stdiod

Path to standard I/O daemon executable. There is one standard I/O daemon per I/O node and it is started by IOSD when the I/O node is booted. If stdio_daemon_path is not specified, the default value is used (/sbin/stdiod).

sysio_daemon_path = /sbin/sysiod

Path to system I/O daemon executable. There is one system I/O daemon for each compute node being serviced by the I/O node. System I/O daemons are started when a compute block is booted and are ended when a compute block is freed. If sysio_daemon_path is not specified, the default value is used (/sbin/sysiod).

toolctl_daemon_path = /sbin/toolctld

Path to tool control daemon executable. There is one tool control daemon for each compute node being serviced by the I/O node. Tool control daemons are started when a compute block is booted and are ended when a compute block is freed. If `toolctl_daemon_path` is not specified, the default value is used (`/sbin/toolctld`).

max_service_restarts = 5

Maximum number of times a service is restarted. IOSD monitors all of the daemons it has started and if a daemon ends abnormally it is automatically restarted. Valid values are 0 to 65535. If `max_service_restarts` is invalid or not specified, the default value is used (5).

JOBCTLD configuration file options

Configuration options used by the job control daemon are in the `[cios.jobctld]` section of the Blue Gene configuration file (`/bgsys/local/etc/bg.properties`). Configurable options include:

listen_port = 7002

The port number used to listen for a Control System connection. If `listen_port` is invalid or not specified, the default port is used (7002).

job_prolog_program_path =

The path to prolog program to run before a job is started. The prolog program must be accessible from I/O nodes. This property is optional and if it is not specified, a prolog program is not run.

job_epilog_program_path =

The path to epilog program to run after a job has ended. The epilog program must be accessible from I/O nodes. This property is optional and if it is not specified an epilog program is not run.

job_prolog_program_timeout = -1

The number of seconds to wait for job prolog program to complete. Valid values are a positive number or -1 to wait forever. If `job_prolog_program_timeout` is invalid or not specified, the default value is used (-1).

job_epilog_program_timeout = -1

The number of seconds to wait for job epilog program to complete. Valid values are a positive number or -1 to wait forever. If `job_epilog_program_timeout` is invalid or not specified, the default value is used (-1).

STDIOD configuration file options

Configuration options used by the standard I/O daemon are in the `[cios.stdiod]` section of the Blue Gene configuration file (`/bgsys/local/etc/bg.properties`). Configurable options include:

listen_port = 7003

The port number used to listen for a Control System connection. If `listen_port` is invalid or not specified, the default port is used (7003).

send_buffer_size = 262144

Size in bytes of send buffer for Control System connection. A larger size allows more standard output and standard error data to be buffered on the I/O node by TCP/IP. If `send_buffer_size` is invalid or not specified, the default value is used (262144).

SYSIOD configuration file options

Configuration options used by the system I/O daemon are in the [cios.sysiod] section of the Blue Gene configuration file (/bgsys/local/etc/bg.properties). Configurable options include:

posix_mode = false

Run I/O operations using Posix rules. When true each I/O operation initiated from a compute node completes atomically. When false, one I/O operation initiated from a compute node might cause multiple I/O operations on the I/O node. When posix_mode is invalid or not specified, the default value is used (false).

log_job_statistics = false

Log system I/O statistics from a job when it ends. When true, information about I/O operations is recorded in the I/O node log file. If log_job_statistics is invalid or not specified, the default value is used (false).

log_function_ship_errors = false

Log information about a function shipped operation when the operation returns an error. When true, the I/O node log file might become large if they are many errors from function shipped operations. If log_function_ship_errors is invalid or not specified, the default value is used (false).

4.7.2 Jobs directory

Information about a job is placed in the /jobs directory on the I/O node. When a job starts, JOBCTLD creates a directory in /jobs using the job identifier. When a job ends, JOBCTLD removes the directory. The job information directory contains the following objects:

exe	A symbolic link to the executable. The value comes from the runjob -exe parameter.
wdir	A symbolic link to the initial working directory. The value comes from the runjob -cwd parameter.
cmdline	A file containing the argument strings. Each string is terminated by a null byte. The end of the strings is denoted by two null bytes. The value comes from the runjob -args parameters.
environ	A file containing the environment variable strings. Each string is terminated by a null byte. The end of the strings is denoted by two null bytes. The value comes from the runjob -envs parameters.
loginuid	A file containing the login user ID value as a string.
loggingids	A file containing the login group ID values as strings. Each string is terminated by a null byte. The end of the strings is denoted by two null bytes.
toolctl_rank	Directory with symbolic links to the tool-control service data-channel local sockets to enable attaching to a specific rank in the job.
toolctl_node	Directory with symbolic links to the tool-control service data-channel local sockets to enable attaching to all ranks in the job.
tools	Directory with symbolic links to the tools that are running. When a tool is started, a symbolic link is added that is named with the tool ID and points to the tool executable path. When a tool ends, its symbolic link is removed.

tools/protocol	A file containing the CDTI version number associated with the Blue Gene system software as a string.
tools/status	Directory with status files for tools that are running. When a tool is started, a zero-length file is added that is named with the tool ID. A tool updates the modification time of its file to tell JOBCTLD that it is still active. When a tool ends, its status file is removed.

4.8 NFS configuration

The Blue Gene/Q system supports Network File System (NFS) v3 and version v4 for all file systems except the root. To set up NFS for Blue Gene/Q on the service node, follow these steps:

1. Configure the number of threads that the NFS server will use in the `/etc/sysconfig/nfs` file. The default for `RPCNFSDCOUNT` is 8. You can adjust the number of threads to suit your environment.
2. 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
```

Export using the following command:

```
exportfs -ar
```

3. Now you can mount `/bgsys` from a client using the following command:

```
mount -o soft 172.16.1.1:/bgsys /bgsys
```

Consider the following points when you perform the configuration of the NFS file server on the service node:

- ▶ The number of NFS server daemons running can be a source of performance issues. If you have too few daemons running, they can become overloaded, which can increase wait times. You can check the use of the `nfsd` threads using the following command:

```
# grep th /proc/net/rpc/nfsd
```

The results look similar to the following line:

```
th 8 594 3733.140 83.850 96.660 0.000 73.510 30.560 16.330 2.380 0.000 2.150
```

In this case, there are 8 `nfsd` threads running, followed by the number of times all of the threads have been needed (594). The 10 numbers that follow show how long, in seconds, a certain fraction of those threads have been busy, starting with less than 10% and ending with over 90%. If the last numbers are high, you can increase the `nfsd` thread count. Increase the number of NFS server daemons by changing the following line in the `/etc/sysconfig/nfs` file:

```
RPCNFSDCOUNT= "xxx"
```

Try using 32, 64, or 128 in place of the `xxx`.

- ▶ If you use the `noac` (no attribute cache) option when mounting NFS directories to avoid problems with multiple users writing to the same file, performance might decrease. 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 example values might reflect extreme conditions. Select values appropriate to your system and memory.

- This setting turns off TCP time stamp 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`
- ▶ Specifying larger values on the NFS mount command for `rsize` and `wsize` might improve performance. Other mount options to consider include:
 - TCP versus UDP mount:
Mounting an NFS share with the UDP protocol on a saturated network can be problematic because you might see a high level of lost packets. Generally, it is best for performance to use TCP only for mounts on an active network.
 - Asynch versus Sync mount:

If you export file systems in NFS with the sync option, general performance might be adversely affected. You might be able to improve overall performance on the clients by using the default asynchronous writes.

The async option allows the server to reply to the client as soon as it processes the request, so that the client does not have to wait for the disk write to complete. The client continues under the assumption that the write completed.

There is a slight risk in using the async option, but only in the situation of a disk failure. The sync option guarantees that all writes are completed when the client thinks they are, but it does so by forcing each client to wait for disk writes to complete.

4.9 Troubleshooting

Output written to the console is sent through the mailbox to the Control System and by default is stored in a log file in the `/bgsys/logs/BGQ` directory. Note that the directory to store log files can be changed by modifying the `log_dir` keyword in the `[mmcs]` section of `bg.properties`. The log file name is the I/O node's location string with `.log` appended to the end (for example, `R00-I5-J06.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.

There is also a `messages` file for each node available under the `/bgsys/linux/nodefs/<node ip>/var/log` directory that is useful when debugging I/O node problems.

4.9.1 NFS errors

This section contains a few of the more common NFS errors and how to debug them.

RPC: Connection Refused

The two main RPC programs that NFS uses are `rpc.mountd` and `rpcbind`. The RPC: Connection Refused error (received from the I/O node) indicates that one of these programs might not be running. You can verify that the programs are running with the following commands on the system running the NFS server (typically the service node):

```
ps -ef | grep rpcbind
```

```
ps -ef | grep rpc.mountd, or rpcinfo -p
```

If you find that `rpcbind` is not running, you can start it using the following command:

```
# /etc/init.d/rpcbind start
```

If both `rpc.mountd` and `nfsd` are not running, you can start both with the following command:

```
# /etc/init.d/nfs start
```

Otherwise, start `rpc.mountd` with the following command:

```
# /usr/sbin/rpc.mountd
```

Also, verify that both programs are configured to start on boot with the following commands:

```
# chkconfig --list rpcbind
```

```
# chkconfig --list nfs
```

If the programs are not configured correctly, set them to start automatically with the following commands:

```
# chkconfig --add nfs
# chkconfig --add rpcbind
```

In some situations, `rpc.mountd` might crash as a result of excessive load. In earlier Linux versions, `rpc.mountd` was a single-threaded, user-level process. You can now pass the `-t` flag with `rpc.mountd` to specify the number of `rpc.mountd` threads to spawn. To increase performance and to prevent `rpc.mountd` from crashing due to load, it is recommended that you take advantage of this option. You must edit the `/etc/init.d/nfs` file. There are two lines in the `nfs` file that can potentially start `rpc.mountd`. Adding the `-t` flag, as shown in Example 4-14, starts `rpc.mountd` with 16 threads.

Example 4-14 Use of the -t flag in the /etc/init.d/nfs file

```
...
if [ -n "$MOUNTD_PORT" ] ; then
startproc /usr/sbin/rpc.mountd -t 16 -p $MOUNTD_PORT
else
startproc /usr/sbin/rpc.mountd -t 16
fi
...
```

RPC: Timeout

This error usually occurs on mount requests when `rpc.mountd` is overloaded. If `rpc.mountd` is overwhelmed, these timeouts occur when running commands, such as `showmount -e <SN IP Address>`, from another Linux system. To lessen the impact of these issues:

- ▶ Space out mount requests to `rpc.mountd`.
- ▶ Verify DNS host resolution of all involved servers. Add the host name and IP addresses for the front end node, service node, and I/O node to the `/etc/hosts` file on the service node. Resolving names and addresses by sending requests to the DNS servers occupies the `rpc.mountd` threads for a longer period of time than if the resolution is done locally. Other mount requests are dropped if they are waiting on a DNS reply. Also, the hosts line in the `/etc/nsswitch.conf` file should be `hosts: files dns` to give preference to `/etc/hosts` over the DNS servers.
- ▶ Increase the number of `rpc.mountd` threads as shown in Example 4-14.

Mount failure: Permission denied

This error is usually generated due to either a missing entry in `/etc/exports`, a syntax error in that file, or due to a mismatch between `/etc/exports` and `/var/lib/nfs/etab`.

Missing entry in /etc/exports

The format of an `/etc/exports` entry is as follows:

```
/filesystem hosts(options)
```

A typical entry for exporting `/bgsys` looks like the entry in Example 4-15.

Example 4-15 Export of /bgsys

```
/bgsys 172.16.0.0/255.255.0.0(rw,async,no_root_squash)
/tmp/gpfsvar
```

```
134.94.72.0/255.255.254.0(rw,no_root_squash,async,fsid=746)
```

After adding an entry to `/etc/exports`, run **exportfs -ar** to make the change known to the NFS daemons. The **exportfs** command makes the change known by modifying `/var/lib/nfs/etab`.

Syntax error in /etc/exports

If there is an unknown option or if the line does not conform to the expected format, the **exportfs** command does not add a line to `/var/lib/nfs/etab` for the file system. Check for any malformed networks, host names, or file system names in the `/etc/exports` file. Also, it is not possible to export both a directory and its child (for example both `/usr` and `/usr/local`). Export the parent directory with the necessary permissions, and then mount all of its subdirectories with those same permissions. After you make any changes, run the **exportfs -ar** command again to make the changes known to the NFS daemons.

Mismatch between /etc/exports and /var/lib/nfs/etab

A *Permission Denied* error on a mount request might simply be the result of someone forgetting to run the **exportfs** command after editing `/etc/exports`. The NFS daemons use the files in `/var/lib/nfs/` as they run. If the file system is in `/etc/exports`, but not in `/var/lib/nfs/etab`, then you need to run the **exportfs -ar** command to make the changes known to the NFS daemons.



5

Control System console

This chapter provides a description of the Blue Gene/Q text-based Control System console.

5.1 Overview

Text-based interaction with Blue Gene/Q is achieved through the **bg_console** command. Communication with the following servers is provided from the console:

- mmcs_server** The Midplane Management Control System (MMCS) is used to configure and allocate blocks of compute nodes and I/O nodes.
- mc_server** The mc_server is a wrapper around the machine controller (mc). The machine controller is a library of functions that provide *Low-Level Control System* (LLCS) access to the hardware.
- runjob_server** The runjob_server facilitates the job launch functions.
- bgmaster_server** BGmaster is the process manager for the Control System servers. Its purpose is to act as a centralized failure and recovery process.
- realtime_server** The real-time server provides real-time notifications of system events to client applications.
- bgws_server** Blue Gene Web Services (BGWS) provides a RESTful web service.

There are two types of commands used with **bg_console**. The first are referred to as *base* commands, and their purpose is to interact with the MMCS server. There is a group of *external* commands that are used to communicate with servers other than the MMCS server. Figure 5-1 provides an overview of the **bg_console** structure.

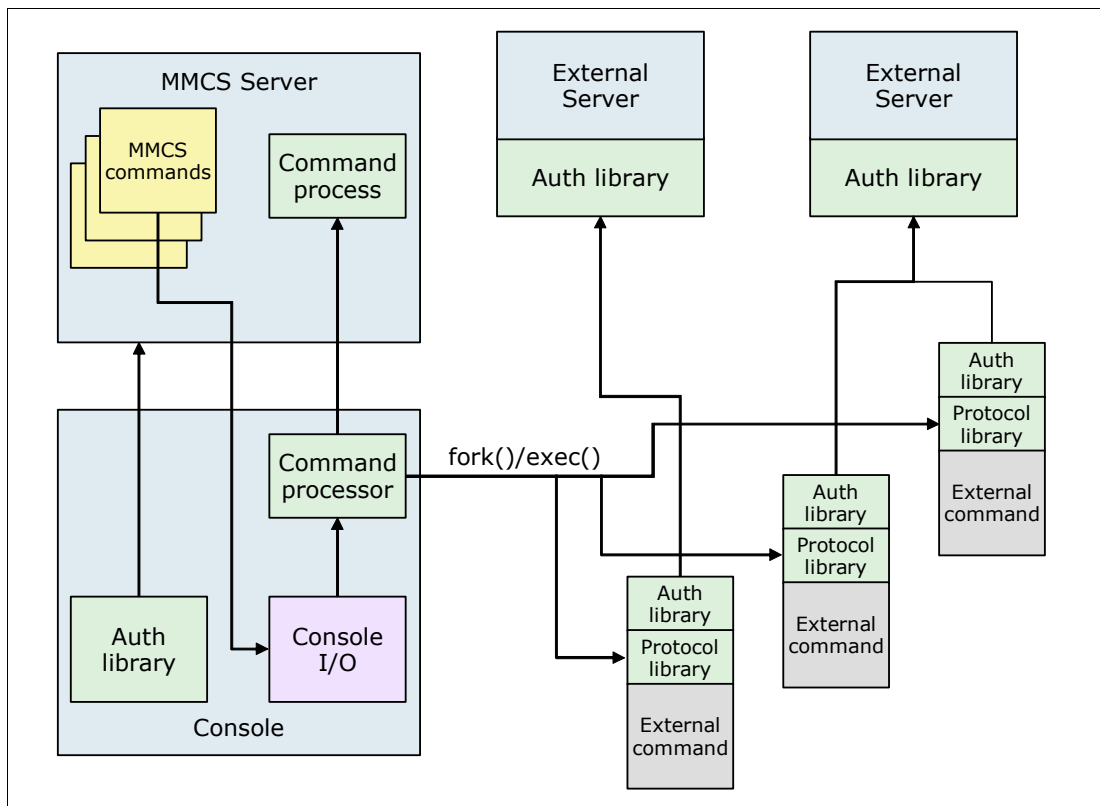


Figure 5-1 *bg_console* overview

The **bg_console** binary is located in the `/bgsys/drivers/ppcfloor/bin` directory. Adding `/bgsys/drivers/ppcfloor/bin` to the users' default path allows access to the console from any

location. One method to add the directory to the default path is to create a script in `/etc/profile.d` that contains the following line:

```
export PATH=$PATH:/bgsys/drivers/ppcfloor/bin
```

5.1.1 Accessing help

The `help` command provides access to the help system. Commands are organized into functional groupings and associated with specific help categories. To get help on commands specific to any group users can type:

```
mmcs$ help <category>
```

Table 5-1 provides a list of the help categories. The category maps to the *Help Category* column in Table 5-2. The `help` command, by itself with no command or category, gives summary help for commands in the default category and also lists the category names.

Table 5-1 Categories available for help command

Category	Description
default	The default set of commands that are read-only, not in any other category, and can be issued by any user.
user	User commands that actually affect something (allocate, for example).
admin	Administrative commands.

To access more information about a specific command use the following syntax:

```
mmcs$ help <command>
```

More information about commands used for servers other than the MMCS server can also be obtained through the command's man page, for example:

```
man list_jobs
```

5.1.2 Command history

The `bg_console` command uses a readline compatible library to provide command-line editing capabilities. Command history is stored in the `$HOME/.bg_console_history` file.

5.2 Commands

Table 5-2 contains the complete list of commands provided through the console by each server. Supported objects and authorities are defined in the Blue Gene/Q security section 18.1, "Object based model" on page 204. The `bgadmin` authority is functionally equivalent to having access to the administrative security certificate.

Table 5-2 bg_console commands

Command	Server	Object/authority	Help category
<code>alias_wait</code>	BGmaster	none	default
<code>binary_status</code>	BGmaster	none	default

Command	Server	Object/authority	Help category
binary_wait	BGmaster	none	default
fail_over	BGmaster	bgadmin	admin
get_errors	BGmaster	none	default
get_history	BGmaster	none	default
monitor_master	BGmaster	none	default
list_agents	BGmaster	none	default
list_clients	BGmaster	none	default
master_start	BGmaster	bgadmin	admin
master_status	BGmaster	none	default
master_stop	BGmaster	bgadmin	admin
delete_block	BGWS server	Block:Delete	user
list_jobs	BGWS server	Job:Read	default
mcservers_clients	MC server	bgadmin	admin
mcservers_status	MC server	Hardware:Read	admin
add_boot_option	MMCS server	Block:Update	admin
allocate	MMCS server	Block:Read/Execute	user
allocate_block	MMCS server	Block:Read/Execute	user
block_status	MMCS server	Block:Read	default
boot_block	MMCS server	Block:Read/Execute	user
connect	MMCS server	Hardware:Read/Execute	admin
copy_block	MMCS server	Block:Create/Read	admin
create_node_config	MMCS server	bgadmin	admin
delete	MMCS server	bgadmin	admin
deselect_block	MMCS server	Block:Read	user
disconnect	MMCS server	Hardware:Read/Execute	admin
dump_block	MMCS server	Block:Read	admin
dump_machine	MMCS server	Hardware:Read	admin
dump_personalities	MMCS server	Block:Read	admin
free	MMCS server	Block:Execute	user
free_all	MMCS server	Block:Execute	user
free_block	MMCS server	Block:Execute	user
gen_block	MMCS server	Block:Create	admin
gen_blocks	MMCS server	Block:Create	admin
gen_full_block	MMCS server	Block:Create	admin

Command	Server	Object/authority	Help category
gen_io_block	MMCS server	Block:Create	admin
gen_midplane_block	MMCS server	Block:Create	admin
gen_small_block	MMCS server	Block:Create	admin
get_block_info	MMCS server	Block:Read	default
get_block_size	MMCS server	Block:Read	default
grant_block_authority	MMCS server	Block owner	user
help	MMCS server	none	default
list	MMCS server	bgadmin	admin
list_blocks	MMCS server	Block:Read	default
list_block_authority	MMCS server	Block:Read	default
list_hw_polling	MMCS server	bgadmin	admin
list_io	MMCS server	bgadmin	admin
list_io_links	MMCS server	Block:Read	admin
list_midplanes	MMCS server	bgadmin	admin
list_selected_block	MMCS server	Block:Read	default
list_users	MMCS server	bgadmin	admin
locate	MMCS server	Block:Read	default
log_level	MMCS server	bgadmin	admin
mmcs_server_connect	MMCS server	none	default
quit	MMCS server	none	default
reboot_nodes	MMCS server	Block:Execute	admin
redirect	MMCS server	Block:Execute	user
redirect_block	MMCS server	Block:Read/Execute	user
revoke_block_authority	MMCS server	Block owner	user
select_block	MMCS server	Block:Execute	default
set_block_info	MMCS server	Block:Update	admin
set_boot_options	MMCS server	Block:Update	admin
show_barrier	MMCS server	Block:Read	admin
sleep	MMCS server	none	default
sql	MMCS server	bgadmin	admin
start_hw_polling	MMCS server	bgadmin	admin
status	MMCS server	bgadmin	admin
stop_hw_polling	MMCS server	bgadmin	admin
sysrq	MMCS server	Block:Execute	admin

Command	Server	Object/authority	Help category
username	MMCS server	none	default
version	MMCS server	none	default
wait_boot	MMCS server	Block:Read	user
write_con or wc	MMCS server	Block:Execute	admin
dump_proctable	runjob server	Job:Read	default
end_tool	runjob server	Job:Execute	user
grant_job_authority	runjob server	Job owner	user
job_status	runjob server	Job:Read	default
kill_job	runjob server	Job:Execute	user
list_job_authority	runjob server	Job owner	default
locate_rank	runjob server	Job:Read	default
revoke_job_authority	runjob server	Job owner	user
start_tool	runjob server	Job:Execute	user
tool_status	runjob server	Job:Read	default
refresh_config	All servers	bgadmin	admin

5.3 Special commands

In addition to the commands described in Table 5-2 on page 69, the console supports the following special commands:

! [*shell command*] This command escapes 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 from the console).

[*<comment>*] This command indicates a comment line.

< *<filename>* This command pipes command input from *<filename>* to MMCS server (can only be used from the console).

5.4 Command parameters

For commands that run against multiple servers, indicated in Table 5-2 on page 69 with *All servers* in the server column, any mandatory parameters will be first and an optional server argument will be the last parameter. If the server argument is not provided, the command will be run against all operational servers. The server implementing the common command will be named *server_command*. The console will set the properties file environment variable to match its properties file:

refresh_config	Re-read bg.properties and apply new or changed options as appropriate.
help	List all commands and their arguments.
version	Provide version information in the format: Driver: [driver] Date: [build date] [optional] The optional string can contain protocol versions or other version identifiers.

5.5 Scripting

Console scripting is available by passing commands to the console's stdin and reading back from stdout within scripts. Example 5-1 shows several examples on how to do this using Perl.

Example 5-1 Scripting console using Perl

```
#!/usr/bin/perl
# This is a set of trivial scripting examples for the bg_console. There are three types:
# 1) Use Perl's expect module to simulate interactive use.
# 2) Execute a single command in the console and have it return.
# 3) Have the console batch-process several commands from a list in a file.

BEGIN {
# This allows access to the Perl expect module necessary for some of these examples.
# If Perl's Expect module is system-installed, then this isn't necessary.
    push @INC, "/perl/lib/perl5/site_perl/5.8.8";
    push @INC, "/perl/lib/perl5/site_perl/5.8.8/ppc-linux-thread-multi";
}

# Use expect Perl module to simulate interactive use:
use Expect;
use warnings;

print "\n****Running Expect script examples****\n";
my $e = Expect->new;

# start bg_console
$e->spawn("./bg_console") or die "Cannot run bg_console\n";

# look for mmcs$ prompt
$e->expect(5, "mmcs\$") or die "no prompt\n";

# send the 'list_blocks' command
$e->send("list_blocks\n");

# wait for an 'OK' to see if it worked
$e->expect(5, "OK") or die "Failed to list_blocks\n";
print "Success! list_blocks\n";

# now send a 'list_users'
$e->send("list_users\n");

# and wait for an 'OK' again
$e->expect(5, "OK") or die "Failed to list_users\n";
print "Success! list_users\n";

print "****Expect scripts done!****\n\n";

# Run a single command and check the result
print "****Running a single command and checking the return****\n";
my $result;
```

```
# Simple. Just echo the command to bg_console's stdin
$result = `echo \"list_users\" | ./bg_console`;

# Now see if we got an 'OK'
if ( $result =~ /OK/ )
{
    print "Success!\n****list_users single command completed****";
}
else
{
    print "Fail!\n****list_users did not succeed****";
}

print "\n\n";

# Run a "batch file" with a list of commands. In this case
# the file is just the following commands in the file named 'commands':
# help all
# list_blocks
# list_users
print "****Now run a batch command file****\n";
my $batch_result;

# This pipes a list of commands into the console's stdin
# The console then executes them serially
$batch_result = `cat commands | ./bg_console`;

$count = 0;
$pos = 0;
$done = 0;

# This just finds all of the 'OK' responses in the result. This isn't really efficient for large batch
# files. You only want to do this kind of thing for single multi-step test cases.
while ( $done == 0 ) {
    $pos = index($batch_result, "OK", $pos);
    if ( $pos == -1 ) {
        $done = 1;
    }
    else {
        $count++;
    }
    $pos++;
}

if ( $count == 3 ) {
    print "****Success! Batch result OK count: ";
    print $count;
    print "****\n";
}
}
```



Submitting jobs

Submitting jobs on Blue Gene/Q is accomplished by using the **runjob** command. The runjob architecture is made up of three components: the runjob server, the multiplexer, and the runjob client. This chapter focuses primarily on the client functions, providing information about:

- ▶ runjob architecture
- ▶ **runjob** command syntax
- ▶ sub-block jobs
- ▶ signal handling
- ▶ standard input, output and error
- ▶ job authority
- ▶ **kill_job** command
- ▶ job status
- ▶ Blue Gene job submission history

6.1 The runjob architecture

The runjob architecture consists of three components: server, multiplexer (mux), and client. The three components are shown in Figure 6-1.

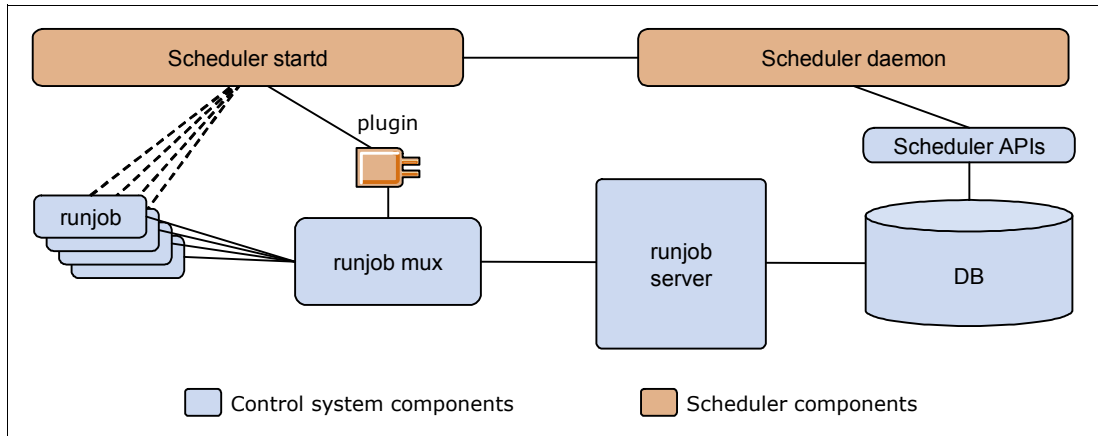


Figure 6-1 The runjob architecture

The purpose of the runjob server is to coordinate job submission. The server inserts, updates, and removes job entries from the database. It communicates with the job control daemon (JOBCTLD) on the I/O nodes to load, start, signal and stop jobs. The runjob server also accepts requests from the runjob mux that were submitted by the clients.

The purpose of the multiplexer is to provide a communication channel between the runjob clients and the runjob server. The multiplexer accepts commands, such as **runjob**, **kill_job** or **job_status**, from one or more local clients and funnels them through a single connection to the server. In a Distributed Control System environment, the multiplexer also facilitates failover. Additionally, the multiplexer provides plug-in functionality to interact with job schedulers.

Note: More detailed information about the runjob server and multiplexer is located in Chapter 14, “The runjob server and runjob mux” on page 163.

The runjob client provides an interface for users to launch and control jobs. The client commands are transmitted to the runjob server through the multiplexer.

Communication between each component is achieved through a request/response protocol.

6.2 The runjob command

The **runjob** command is the interface to launch jobs on Blue Gene/Q. It acts as a shadow process for the job's lifetime. The syntax of the command is:

```
runjob [OPTIONS ... ] : exe [arg1 arg2 ... argn]
```

The block that a job is intended to run on must be booted prior to the **runjob** command being issued. If the block that is requested to run a job is not initialized, **runjob** immediately terminates with a descriptive error message.

The command-line behavior can be modified when using a job scheduler, for example, if the `--block` option is omitted from the command, an external scheduler can decide which block to use. More information about the Scheduler APIs is in the online documentation that is shipped with Blue Gene/Q in the `/bgsys/drivers/ppcfloor/hlcs/docs/bgsched/external` directory. You can also display the Scheduler API documentation from the Knowledge Center in the Navigator.

6.2.1 The runjob options

Table 6-1 provides a list of supported arguments for the `runjob` command. All arguments can also be specified by environment variables by prefixing `RUNJOB_` to their name. For example, the `--exe` argument can be specified with `RUNJOB_EXE`. Command-line arguments always take precedence over their environmental equivalent.

Table 6-1 `runjob` options

Argument	Description
Job options	
<code>--exe</code>	<p>Path name for the executable to run. Your user ID must have permission to read and execute this executable. The path can be an absolute path or a relative path to <code>--cwd</code>. This value can also be specified as the first positional parameter after the <code>:"</code> token. Note the total length of all <code>--args</code> and <code>--exe</code> arguments must be less than 4097 characters. The executable must be less than 513 characters.</p> <p>The <code>--exe</code> parameter is required.</p> <p>For example: runjob --block R00-M0 : /home/user/exe With the executable specified as the first positional argument, is equivalent to: runjob --block R00-M0 --exe /home/user/exe</p>
<code>--args</code>	<p>Arguments for the executable specified by <code>--exe</code>. Multiple tokens to <code>--args</code> are split on spaces until another argument is encountered. To retain spaces in your tokens, enclose them in quotation marks. Multiple <code>--args</code> parameters must be used to differentiate between <code>runjob</code> arguments and application arguments. Arguments can also be specified after the first positional executable parameter, in which case differentiation between <code>runjob</code> arguments and application arguments is not necessary.</p> <p>Arguments are composing, their values are combined from both command-line parameters and environment variables (<code>RUNJOB_ARGS</code>) to generate the final set. Note that all <code>--exe</code> and <code>--args</code> parameters must be less than 4097 characters.</p> <p>For example: runjob --block R00-M0 : a.out hello world is equivalent to: runjob --block R00-M0 --args hello world --exe a.out And: runjob --block R00-M0 --args --one --args --two --exe a.out is equivalent to: runjob --block R00-M0 : a.out --one --two</p>

Argument	Description
--envs	Environment variables to export. This value must contain an equals character that is not the first character of the string. Multiple tokens are split on spaces until another argument is encountered. The leading character of an environment variable name cannot be a digit or a space. To retain spaces in your environment variables enclose them in double quotes. Environment variables are composing: their values are combined from both arguments and environment variables to generate the final set. The total length of all environment variables must be less than 8192 characters, which includes the environment variable name and the equals character.
--exp_env	Export an environment variable from the current environment to the job. Multiple tokens are split on spaces. The variable must be present in the current environment.
--env_all	Export all environment variables from the current environment to the job. This can quickly increase the total size of environment variables past the 8192 character limit. Use this argument with caution.
--cwd	The current working directory for the job. This value must be a fully qualified path. The default value is the working directory when runjob is started. Must be less than 512 characters.
--timeout	After the job starts running, deliver a SIGKILL after the specified number of seconds.
Resource options	
--block	The compute block ID to run the job on. The block must be initialized prior to invoking runjob. If the system administrator has configured a job scheduler, this parameter can be changed before the job starts. This parameter must be less than 33 characters. Also see --corner and --shape for requesting that a job use sub-block resources.
--corner	Location string of a single node or single core within the block to represent the corner of a sub-block job. If the system administrator configured a job scheduler, this parameter can be changed before the job starts. For example: <ul style="list-style-type: none"> ▶ R00-M0-N00-J00 (node J00 on board N00 in midplane R00-M0) ▶ R00-M1-N12-J05-C00 (core 0 on node J05 on board N12 in midplane R00-M1) Core corner locations cannot use a shape or alternate ranks per node; therefore, --shape and --ranks-per-node are both ignored when specified with a core corner location. Node corner locations require a --shape argument. Multiple sub-node jobs on a single node using core corner locations are limited to a single user. The --corner argument requires that --block be also be specified.
--shape	Five dimensional AxBxCxDxE torus shape of a sub-block job, starting from a corner given by --corner. Must be equal to or less than a midplane (4x4x4x4x2) in size and cannot span multiple midplanes. Shapes also cannot wrap around torus dimensions. All dimensions must be a power of two: either 1, 2, or 4 compute nodes in size. If the system administrator has configured a job scheduler, this parameter can be changed before the job starts. The --shape argument requires that --corner and --block be specified as well.
--ranks-per-node	Number of processes per compute node. Valid values are 1, 2, 4, 8, 16, 32, and 64.

Argument	Description
--np	Number of processes in the entire job. The default value is the entire size of the block or sub-block shape. The --np value must be equal to or less than the block (or sub-block shape) size multiplied by the ranks per node value.
--mapping	Permutation of ABCDET or a path to a mapping file containing coordinates for each process. If the path is relative, the combination of --cwd and --mapping must be less than 513 characters. If the path is absolute, --mapping must be less than 513 characters. The syntax of a mapping file is 6 columns separated by white space. The columns indicate the A,B,C,D,E,T coordinates, and the line number is the process. Comments can be used anywhere in the line with the # character, any text after the # is ignored. An example of mapping file contents: <pre> 0 0 0 0 0 # rank 0 0 0 0 1 0 # rank 1 0 0 1 0 0 # rank 2 0 0 1 1 0 # rank 3 </pre>
Debug options	
--label	Enable or disable standard output and standard error prefix labels. Values are none, short, or long. None omits all prefixes. Short includes only the rank. Long includes the source in addition to the rank in square brackets. The default value is none, the implicit value is long. Examples: <ul style="list-style-type: none"> ▶ --label short 1: hello world ▶ --label long stdout[1]: hello world ▶ --label stdout[1]: hello world ▶ --label none hello world
--strace	Enable or disable system call tracing. Values are none or a specific rank to enable the tracing. The rank must be a valid rank in the job.
--start-tool	Fully-qualified path to the tool daemon to launch before the job starts. The combination of this value and --tool-args must be less than 4097 characters. Tools cannot be started for sub-node jobs. When using a mapping file (see --mapping) this file must be readable by the runjob_server (typically the bgqadmin uid) to correctly calculate location to rank mappings.
--tool-args	Single token containing arguments to be passed to the tool daemons. This token is split on spaces. The --start-tool option must be given with this option. The combination of this value and --start-tool must be less than 4097 characters.
--tool-subset	A subset specification is a single token that consists of a space separated list. Each element in the list can have one of three formats: <ul style="list-style-type: none"> ▶ rank1 : single rank ▶ rank1-rank2 : all ranks between rank1 and rank2 ▶ rank1-rank2:stride : every strideth rank between rank1 and rank2 A rank specification can either be a number or the special keywords \$max or max. Both of them represent the last rank participating in the job. The \$ character is optional to prevent shell escaping. This subset specification will restrict the I/O nodes used for launching the tool. Note each element in the specification must be in increasing order, and cannot have any overlapping ranks. The subset must specify at least one rank that is participating in the job. The --start-tool option must be given with this option. If not specified, the default value is 0-\$max.

Argument	Description
Miscellaneous options	
--stdinrank	The rank to deliver standard input to. Must be a valid rank in the job.
--raise	If the job abnormally terminates with a signal, re-raise that signal when runjob terminates.
-h [--help]	Displays the help text for the runjob command.
-v [--version]	Displays version information
--properties	Location of the bg.properties configuration file to be used if other than the default. The default location is in the /bgsys/local/etc directory.
--verbose	Sets the logging configuration. Setting these values overrides the logging configuration in the bg.properties file. The --verbose option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level; OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the ibm.utility logger to the debug value, the following command can be used: runjob --verbose=DEBUG --verbose=ibm.utility=DEBUG

6.2.2 Scheduler

Blue Gene/Q job submission usually occurs through a scheduler that typically wraps the **runjob** command. The Blue Gene/Q solution provides interfaces for external job schedulers (resource managers) to access system hardware topology and to manage blocks and jobs. Overall, these interfaces are known as the Scheduler APIs, which are documented in the online documentation shipped with Blue Gene/Q in the /bgsys/drivers/ppcfloor/hlcs/docs/bgsched/external directory. You can also display the Scheduler API documentation from the Knowledge Center in the Navigator.

Job schedulers can prevent users from submitting jobs using the **runjob** command. Check documentation of the scheduler system for usage information.

6.3 Sub-block jobs

Sub-block jobs on Blue Gene/Q are jobs that coexist with each other on a single compute block. You can run single node (non-MPI) style jobs and multiple MPI jobs on the same block. These jobs share the I/O nodes of the block but run on different compute node cores.

Sub-block jobs are intended to provide better utilization of compute resources so that you can reduce the number of idle compute nodes and pack jobs more tightly onto the system. For example, if sub-block jobs support is not used, and you have a job that only needs 10 compute nodes, you must allocate, at minimum, a 32 compute node block (assuming the best I/O cabling) and waste 22 of the compute nodes. Using the sub-block job software feature you can create that same 32 compute node block and run the 10 compute node job and other sub-block jobs on the same block.

You can have a sub-block job running on a set of compute nodes that has no I/O connection. This is one of the primary purposes of sub-block jobs, to enable more jobs than there are I/O connections. So using sub-block jobs, you eliminate the restrictions on number of simultaneous jobs that would have been imposed by the number of I/O nodes.

The sub-block job capability is enabled by the dynamic job configuration performed by CNK and the system class routes calculated at job launch time rather than block boot time. A sub-block job differs from a job using the entire block by requesting a compute-node corner location and a five-dimensional shape at job launch time. This corner and shape are combined to logically create a sub-block, which is then used for resource arbitration to ensure overlapping compute nodes are not in use by active jobs. It is also used to enforce that the job is entirely contained within the corner compute node's midplane. Using a shape effectively enforces a collection of compute nodes without any holes, easing the challenges this would otherwise pose for the messaging communications software stack. The shape of a sub-block job has a maximum size of 512 compute nodes. This limitation is solely imposed by software and not due to a hardware limitation. The logic behind this restriction is a 512 node midplane, which is the building block for creating larger blocks. Doing so also greatly simplifies the resource arbitration. Any shape, between a single node (1x1x1x1x1) and the entire midplane (4x4x4x4x2), with all dimensions being a power of 2, is a valid shape for a sub-block job. Most importantly, job sizes are not limited by any artificial I/O ratio.

Figure 6-2 shows a two-dimensional layout of six sub-block jobs using a variety of corner and shape combinations. It is important to note that the compute nodes in the empty space between the logical layout of the jobs are available for use. They are not orphaned, reserved, or idled by another job.

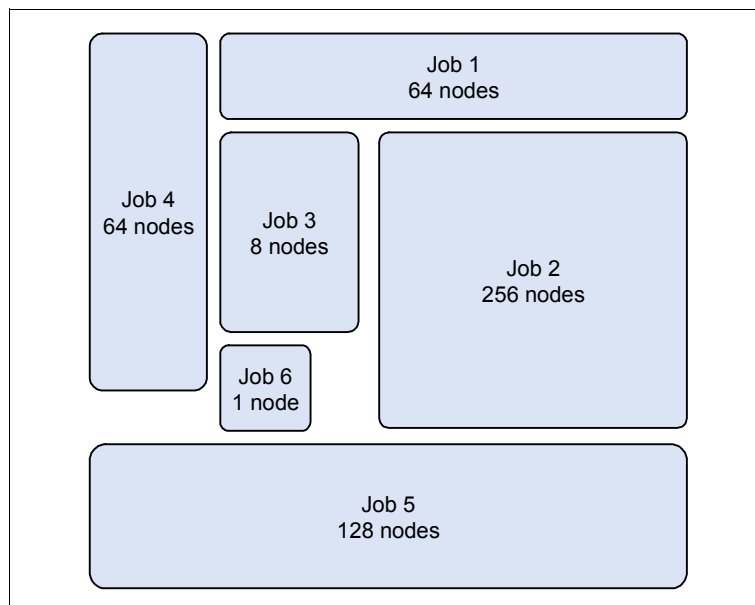


Figure 6-2 Blue Gene/Q sub-block jobs

There are several application considerations when using sub-block jobs. Foremost, the I/O resources are shared with other jobs serviced by the same I/O node. Considering the six jobs shown in Figure 6-2, one job can monopolize resources on the I/O node. If any of the five remaining jobs need guaranteed I/O bandwidth, precautions might need to be taken by the scheduler to ensure that adequate resources are available. Sub-block jobs do not have the same electrical isolation guarantee that their full-block job counterparts do. On all Blue Gene/Q systems, a block of compute nodes is electrically isolated from neighboring compute nodes when it is booted. Because sub-block jobs are sharing the same compute node to I/O node bridge, this guarantee cannot be enforced. This can be a security concern to certain applications if they have a requirement to not share the torus links between compute nodes.

6.4 Signal handling

SIGINT and SIGXCPU signals are handled by the **runjob** command. Delivering either one causes **runjob** to deliver a SIGKILL to the job.

Signals other than SIGINT or SIGXCPU are not handled by **runjob**. They can be delivered to the job using the **kill_job** command. The default action is taken for signals other than SIGINT or SIGXCPU. Depending on the signal, this might be to terminate the process or ignore the signal. Similarly, the kill timeout can be changed when using **kill_job** instead of delivering a signal to **runjob**. Successive signals sent to **runjob** will not cause additional SIGKILL signals to be delivered to the job. Only the first one causes the kill timer to start.

6.4.1 Exit status

If the job fails to start for whatever reason, the exit status will be EXIT_FAILURE(1), and an informative message is logged that indicates the failure. After a job starts, the exit status is the first non-zero exit status reported by any process in the job. For non-zero exit status values, a message is logged that indicates which process caused the termination. When a process terminates with an unhandled signal, **runjob** can optionally re-raise this signal using the --raise option. If --raise is not given, the exit status will be 128 + the signal number. An informative message is also logged indicating which process generated the signal.

If delivering a SIGKILL times out, the exit status will be EXIT_FAILURE, and an informative message will be logged. Example 6-1 provides some sample messages.

Example 6-1 Exit status messages

```
2011-04-25 16:13:23.711 (WARN ) [0xffff8b015750] R00-M0-N00:40:ibm.runjob.client.Job: normal termination with status 15 from rank 0
2011-04-25 16:15:08.415 (WARN ) [0xffffafa75750] R00-M0-N00:41:ibm.runjob.client.Job: terminated by signal 15
2011-04-25 16:15:08.415 (WARN ) [0xffffafa75750] R00-M0-N00:41:ibm.runjob.client.Job: abnormal termination by signal 15 from rank 0
2011-04-25 16:14:10.021 (ERROR) [0xffff7a6b5750] R00-M0-N00:5388:ibm.runjob.client.MuxConnection: could not connect: Connection refused
2011-04-25 16:14:29.084 (FATAL) [0xffffaacd5750] R00-M0-N00:5424:ibm.runjob.client.Job: could not start job: runjob_server is unavailable
```

6.4.2 Normal job termination

If any process in the job terminates normally with exit(1), the remaining processes are delivered SIGTERM.

6.4.3 Abnormal job termination

If any process in the job abnormally terminates by a signal, the remaining processes are delivered a SIGKILL.

The **runjob** command uses a local (AF_UNIX) socket to the runjob_mux process. If this connection is prematurely closed due to the process abnormally terminating, the job is delivered a SIGKILL signal.

6.5 Standard input, output and error

After a job starts, standard input, output, and error are transparently forwarded by the **runjob** command on behalf of the application. Each line of output or error can be optionally prefixed with the rank using the `--label` option. Note that input is only forwarded to the application when requested. In other words

```
read( STDIN_FILENO, buf, 1024 );
```

by the rank specified using `--stdinrank` triggers a corresponding read system call by **runjob**. For ranks other than `--stdinrank`, a read system call will always return 0. Note that when the standard input file descriptor for the **runjob** command is a tty, line buffering is enabled. Otherwise, block buffering is used.

6.6 Job authority

A user must have authority to perform an action on a job (see Chapter 18, “Security” on page 203 for a detailed discussion). The owner of the job has all authority to the job and can grant authority to another user before that user can perform an action on the owner’s job. There are three commands that relate to job authority: **grant_job_authority**, **revoke_job_authority** and **list_job_authority**. These commands are located in the `/bgsys/drivers/ppcfloor/bin` directory.

6.6.1 The `grant_job_authority` command

The **grant_job_authority** command grants another user permission to read or execute privileges on a job. Granting authority to a job does not affect special authorities created in the `bg.properties` file. The syntax of the command is:

```
grant_job_authority [OPTIONS ... ]
```

Table 6-2 shows the options that are available for the **grant_job_authority** command.

Table 6-2 Job authority command options

Option	Description
<code>--id</code>	Numeric job ID. This can be specified as the first positional argument as well.
<code>--user</code>	Specifies the user or group name to grant or revoke authority. This can be specified as the second positional argument as well.
<code>--action</code>	Specifies the action being granted or revoked (read or execute). This can be specified as the third positional argument as well.
<code>--version</code>	Displays the version information for the grant_job_authority or revoke_job_authority command.
<code>--properties</code>	Location of the <code>bg.properties</code> file to be used if other than the default. The default location is in the <code>/bgsys/local/etc</code> directory.

Option	Description
--verbose	Sets the logging configuration. Setting these values overrides the logging configuration in the <code>bg.properties</code> file. The <code>--verbose</code> option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level; OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the <code>ibm.utility</code> logger to the debug value, the following command could be used: <code>grant_job_authority --verbose=DEBUG --verbose=ibm.utility=DEBUG</code>
--host	Server host and port to connect to. Enclose IPv6 addresses in square brackets. For link-local addresses, append the interface to the address after a “%” character. Specify the port after the interface using a colon. The following values are examples values: <ul style="list-style-type: none"> ▶ 127.0.0.1 ▶ 172.16.3.1:12345 ▶ [::1] ▶ [fe80::214:5eff:fre9c:52ce%eth0] ▶ [::1]:24555
--wait-for-server	Keep trying to connect to the server if it is unavailable.
--help	Display help text for the <code>grant_job_authority</code> or <code>revoke_job_authority</code> command.

6.6.2 The revoke_job_authority command

The `revoke_job_authority` command removes the read or execute privileges from the user and job specified in the command. Revoking authority to a job does not affect special authorities created in the `bg.properties` file. The syntax of the command is:

```
revoke_job_authority [OPTIONS ... ]
```

The options available for the `revoke_job_authority` command are the same as the `grant_job_authority` command. Refer to Table 6-2 on page 83 for specific options.

6.6.3 The list_job_authority command

The `list_job_authority` command returns a list of users and groups that have authority to perform various actions on the job specified in the command. The syntax of the command is:

```
list_job_authority [OPTIONS ... ]
```

The options available for the `list_job_authority` command are provided in Table 6-3.

Table 6-3 list_job_authority options

Option	Description
--id	Numeric job ID. This can be specified as the first positional argument as well.

Option	Description
--host	Server host and port to connect to. Enclose IPv6 addresses in square brackets. For link-local addresses, append the interface to the address after a “%” character. Specify the port after the interface using a colon. The following values are examples values: <ul style="list-style-type: none"> ▶ 127.0.0.1 ▶ 172.16.3.1:12345 ▶ [::1] ▶ [fe80::214:5eff:fre9c:52ce%eth0] ▶ [::1]:24555
--version	Display the version information for the <code>list_job_authority</code> command.
--properties	Location of the <code>bg.properties</code> file to be used if other than the default. The default location is in the <code>/bgsys/local/etc</code> directory.
--verbose	Sets the logging configuration. Setting these values overrides the logging configuration in the <code>bg.properties</code> file. The <code>--verbose</code> option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level; OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the <code>ibm.utility</code> logger to the debug value, the following command could be used: <code>list_job_authority --verbose=DEBUG --verbose=ibm.utility=DEBUG</code>
--wait-for-server	Keep trying to connect to the server if it is unavailable.
--help	Display help text for the <code>list_job_authority</code> command.

6.7 The `kill_job` command

The `kill_job` command delivers a signal to a job asynchronously. Signals other than SIGKILL can only be sent to a running job. The syntax of the `kill_job` command is:

```
kill_job [OPTIONS ... ]
```

The options available for `kill_job` are shown in Table 6-4.

Table 6-4 `kill_job` options

Option	Description
--id	Numeric job ID. This can be specified as the first positional argument as well.
--signal -s	Signal to send. This can be specified as an integer or the name of the signal. The <code>--signal</code> and <code>-s</code> switch is optional; instead, the signal number or string can be prefixed by a single <code>-</code> character similar to Linux's <code>kill</code> command. If no signal is provided, SIGKILL is sent by default.
--timeout	The number of seconds to start a timeout when delivering SIGKILL. If the job has not finished on its own after this value, it is forcefully terminated and the nodes in use are unavailable for future jobs until the block is rebooted. This value is ignored for signals other than SIGKILL.
--version	Display the version information for the <code>kill_job</code> command.
--properties	Location of the <code>bg.properties</code> file to be used if other than the default. The default location is in the <code>/bgsys/local/etc</code> directory.

Option	Description
--verbose	Sets the logging configuration. Setting these values overrides the logging configuration in the <code>bg.properties</code> file. The <code>--verbose</code> option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level; OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the <code>ibm.utility</code> logger to the debug value, the following command could be used: <code>kill_job --verbose=DEBUG --verbose=ibm.utility=DEBUG</code>
--host	Server host and port to connect to. Enclose IPv6 addresses in square brackets. For link-local addresses, append the interface to the address after a “%” character. Specify the port after the interface using a colon. The following values are examples values: <ul style="list-style-type: none"> ▶ 127.0.0.1 ▶ 172.16.3.1:12345 ▶ [::1] ▶ [fe80::214:5eff:fre9c:52ce%eth0] ▶ [::1]:24555
--wait-for-server	Keep trying to connect to the server if it is unavailable.
--help	Display help text for the <code>kill_job</code> command.

Examples of using the `kill_job` command are:

To send a SIGKILL to the job numbered 456, use the following command:

```
kill_job 456
```

To send a SIGTERM to job number 123, use the following command:

```
kill_job -TERM 123
```

6.8 Job status

The status of a job changes as it transitions from one state to another. Figure 6-3 shows the possible states of a job.

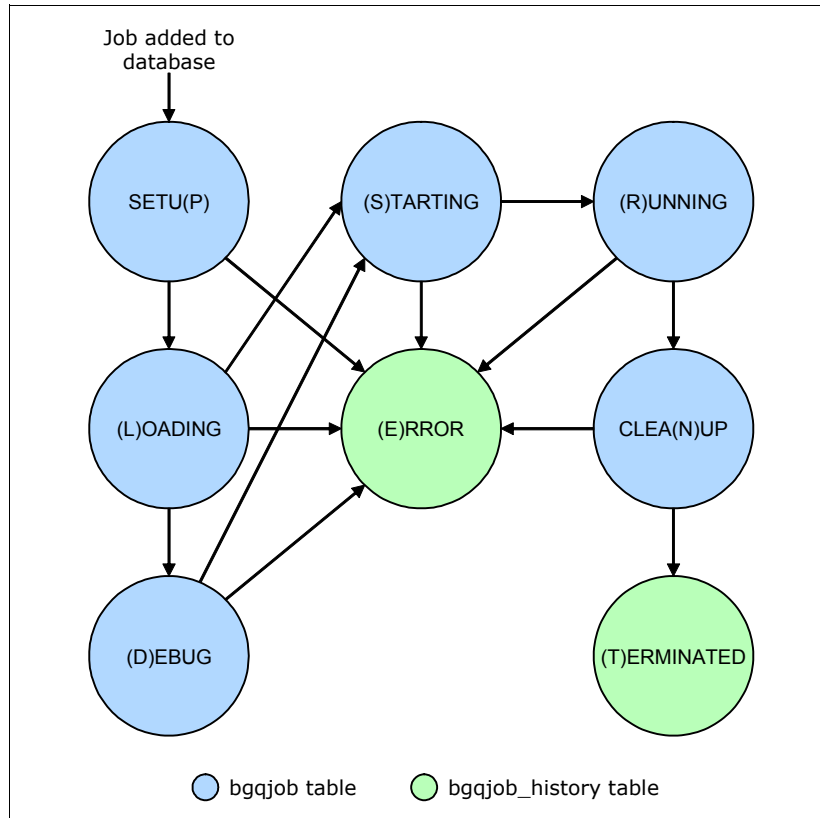


Figure 6-3 Job status transitions

The following list provides more details about each of the possible states:

Setu(P)

In this step, the collective and global interrupt class route settings are calculated for each node in the job. This takes the `--np` and `--ranks-per-node` settings into account to properly create a circumscribing logical rectangle for all the nodes in use. A logical root of the rectangle is selected to act as the job leader. This node is responsible for coordinating abnormal and normal termination for each rank in the job.

(L)oading

In this step, the job description (executable, arguments, environment) is communicated to the I/O nodes in use by the job. Permission to read and execute the executable is validated and the compute nodes load the executable from the file system through their I/O node.

(D)ebug

This status occurs between the Loading and Starting statuses when the command `runjob --start_tool` is used.

(S)tarting

After all of the compute nodes have loaded the executable, they are all told to release control to the application so it can start running.

(R)unning

After all compute nodes have reported starting the application, the job has Running status. It retains this status until it terminates normally or abnormally.

Clea(N) Up

For normal termination, a job has a Cleanup status while the resources on the I/O nodes and compute nodes are cleaned up in preparation for future jobs.

(E)rror

A job has this status in the history table when it terminates without entering the Running stage or if delivering a SIGKILL timed out.

(T)erminated A job has this status in the history table when it completes. The exit status and error text columns will have more information about how the job completed.

Job status can be viewed with Navigator, using the `job_status` command or queried directly from the database.

6.8.1 The `list_jobs` command

The `list_jobs` command is issued in the `bg_console` environment. Running the `list_jobs` command without options returns a list of all the active jobs currently on the system that the user has access to. The syntax of the command is:

```
list_jobs [OPTIONS ... ]
```

Table 6-5 provides the supported options for the `list_jobs` command.

Table 6-5 `list_jobs` options

Option	Description
-l [--long]	Print the summary using long format.
-a [--all]	Fetch all jobs rather than limit.
-h [--help]	Print help text.
--verbose	Sets the logging configuration. Setting these values overrides the logging configuration in the <code>bg.properties</code> file. The <code>--verbose</code> option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level; OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the <code>ibm.utility</code> logger to the debug value, the following command could be used: list_jobs --verbose=DEBUG --verbose=ibm.utility=DEBUG
Job summary filter and sort options	
--status	Supply the job status to use as filter(s). Possible values are D, E, L, N, P, R, S, T or "all". The default is current jobs (DLNPRS). See 6.8, "Job status" on page 86 for a description of each of the job statuses.
--block	The block ID
--executable	The name of the executable file
--user	The user name that submitted the job
--start-time	Start time of the job using the format: <i>interval=yyyymmddThhmmss/yyyymmddThhmmss</i>
--end-time	End time of the job using the format: <i>interval=yyyymmddThhmmss/yyyymmddThhmmss</i>
--exit-status	Job exit status
--sort	Sort field and direction

Example 6-2 shows the output of the `list_jobs` command with no options specified.

Example 6-2 `list_jobs` output with no options

```
mmcs$ list_jobs
```

```

OK
2 jobs
  ID  Status  exe          block        user
142734 R      bgqmpi.elf  R00-M1      bgquser1
142776 R      split      R00-M0-N03  bgquser2

```

Example 6-3 shows the `list_jobs` output when the `-l` option is specified.

Example 6-3 list_jobs output with -l option

```

mmcs$ list_jobs -l
OK
ID: 142734
User: 'bgquser1'
Executable: 'bgqmpi.elf'
Status: Running
Block: R00-M1
Nodes: 512
Processes per node: 1
Start time: 2011-05-02 13:33:10.555433

ID: 142776
User: 'bgquser2'
Executable: 'split'
Status: Running
Block: R00-M0-N03
Nodes: 32
Processes per node: 16
Start time: 2011-05-02 13:44:02.387233

```

6.8.2 The `job_status` command

The `job_status` command can also be used from the command line to obtain information about a specific job. The syntax of the command is:

```
job_status [OPTIONS ... ]
```

Table 6-6 provides a description of the available options for the `job_status` command.

Table 6-6 job_status command options

Option	Description
<code>--id</code>	Job ID, required parameter.
<code>--version</code>	Display the version information for the <code>job_status</code> command.
<code>--properties</code>	Location of the <code>bg.properties</code> file to be used if other than the default. The default location is in the <code>/bgsys/local/etc</code> directory.
<code>--verbose</code>	Sets the logging configuration. Setting these values overrides the logging configuration in the <code>bg.properties</code> file. The <code>--verbose</code> option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level; OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the <code>ibm.utility</code> logger to the debug value, the following command could be used: job_status --verbose=DEBUG --verbose=ibm.utility=DEBUG

Option	Description
--host	Server host and port to connect to. Enclose IPv6 addresses in square brackets. For link-local addresses, append the interface to the address after a “%” character. Specify the port after the interface using a colon. The following values are examples values: <ul style="list-style-type: none"> ▶ 127.0.0.1 ▶ 172.16.3.1:12345 ▶ [::1] ▶ [fe80::214:5eff:fre9c:52ce%eth0] ▶ [::1]:24555
--wait-for-server	Keep trying to connect to the server if it is unavailable. This option takes one argument which is the number of seconds to wait before retrying to connect.
--help	Display help text for the <code>job_status</code> command.

Example 6-4 shows a sample of the output generated by the `job_status` command.

Example 6-4 job_status output

```
user@bgq ~> job_status 43
2 I/O connections
Location      Compute Nodes  Drained  Dying  Ended  Error  Exited  Loaded  Output  Running
R00-ID-J02           16
R00-ID-J03           16
user@bgq ~>
```

6.9 Historical perspective of job submission on Blue Gene

The `runjob` command replaces numerous methods for job submission from the previous Blue Gene models into a single interface. In Blue Gene/P, the following methods can all be used to submit a job:

- ▶ `mpirun` for MPI jobs
- ▶ `submit` for single node HTC jobs
- ▶ `submit_job` from a `mmcs_db_console`

The `runjob` command is conceptually the same as the `mpirun` command from previous Blue Gene software releases because it acts as a shadow process for the job's lifetime. It is important to note that `runjob` has one significant difference compared to `mpirun` in that it is not a scheduler. It also differs in the fact that it does not provide any queuing support nor any support for creating or booting blocks.

Figure 6-4 shows the job submission architecture in Blue Gene/P using `mpirun`.

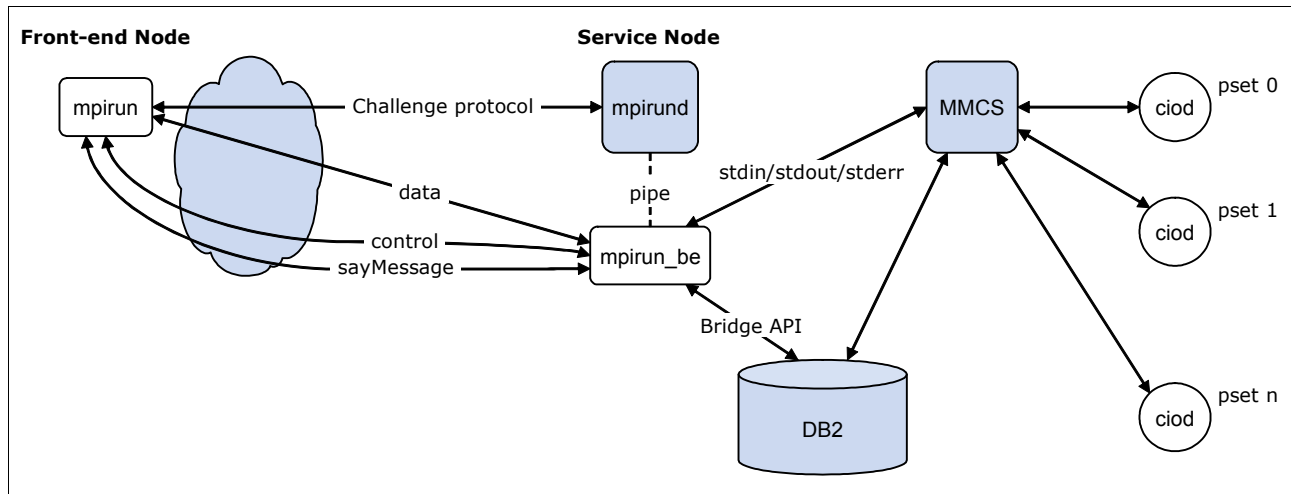


Figure 6-4 Blue Gene/P job submission architecture

The runjob architecture is noticeably different. Foremost, runjob uses a client/server architecture rather than a front-end/back-end design like `mpirun`. From a high level, the mapping of component roles can be loosely interpreted as follows:

- ▶ `mpirun` ∅ client
- ▶ `mpirun_be` ∅ mux
- ▶ `mpirund` ∅ server
- ▶ `mmcs` ∅ server

The challenge protocol used between `mpirun` and `mpirund` was replaced by a local socket used between the runjob client and mux and an SSL connection between the mux and the server. The runjob mux does not interact with the database like `mpirun_be` did, primarily because it does not use the database as an interprocess communication (IPC) mechanism.

The most noticeable difference of `runjob` compared to `mpirun` is its inability to create and boot blocks. The `runjob` command requires the block to be already initialized prior to its invocation.

Another important difference between `mpirun` and `runjob` is the command-line argument processing. Experienced `mpirun` users can launch a job as follows:

```
mpirun -partition R00-M0 -exe /home/user/hello
```

Whereas the equivalent `runjob` invocation is:

```
runjob --block R00-M0 --exe /home/user/hello
```

You will notice that `runjob` requires two dashes instead of one before each argument. The vast majority of arguments used to describe a job are similar between `mpirun`, `submit`, and `runjob`, consult the `runjob` command documentation in section 6.2, "The runjob command" on page 76 for in-depth explanation of the arguments and their supported values.

Table 6-7 shows a comparison of commands in Blue Gene/P and the Blue Gene/Q equivalent.

Table 6-7 Command comparison

Blue Gene/P mpirun	Blue Gene/P submit	Blue Gene/Q runjob	Notes
-exe	-exe	--exe	
-args	-args	--args	Multiple tokens instead of a single token enclosed with double quotation marks
-env	-env	--envs	Multiple tokens allowed
-exp_env	-exp_env	--exp-env	Multiple tokens allowed
-env_all	-env_all	--env-all	
-partition	-pool	--block	
-mapping or -mapfile		--mapping	
-np		--np	
-mode	-mode	--ranks-per-node	Syntax has changed
	-location	--corner	
-shape		--shape	Used for sub-block jobs instead of block creation
-strace	-strace	--strace	
-start_gdbserver	-start_gdbserver	--start-tool	--tool-args to pass arguments
	-raise	--raise	
-label		--label	
-timeout	-timeout	--timeout	
-psets_per_bp			No longer supported.
-noallocate			No longer supported.
-free			No longer supported.
-nofree			No longer supported.
-connect			No longer supported.
-host		--socket	UNIX socket instead of TCP
-port	-port	--socket	UNIX socket instead of TCP
-verbose		--verbose	Syntax differs noticeably, see 6.2.1, "The runjob options" on page 77
-trace	-trace	--verbose	Syntax differs noticeably, see 6.2.1, "The runjob options" on page 77
-config	-config	--properties	



Reliability, availability, and serviceability

The reliability, availability, and serviceability (RAS) infrastructure provides the means to report events that occur on the hardware and in the software on the Blue Gene/Q system. A predefined set of messages are built in, but the design also allows users to define their own RAS events and log them to the database. RAS events can be viewed and queried through Blue Gene Navigator (See section 2.2.8, “RAS” on page 17).

7.1 Elements of a RAS message

Blue Gene/Q RAS messages contain basic information, such as a message ID, severity, location, and text, related to the event. RAS elements that make up the messages are defined in the `/bgsys/drivers/ppcfloor/ras/include/ras.xml` file. This section describes the following elements of RAS messages:

- ▶ Message ID and component
- ▶ Category
- ▶ Severity
- ▶ Control Action
- ▶ Diagnostics

7.1.1 Message ID and component

The message ID is the unique identifier of a RAS event. The ID is made up of an eight-character hexadecimal number. The component is the software unit detecting and reporting the RAS event and can be up to 16 characters long. Each component is assigned a specific range of message IDs to avoid conflict.

Table 7-1 shows the default components and their assigned range of message IDs as defined in the RAS definition file.

Table 7-1 RAS components and message range

Component	Start ID number	End ID number
RESERVED1	00000000	0000FFFF
CNK	00010000	0001FFFF
DIAGS	00020000	0002FFFF
BGMASTER	00030000	0003FFFF
MC	00040000	0004FFFF
MCSERVER	00050000	0005FFFF
MMCS	00060000	0006FFFF
BAREMETAL	00070000	0007FFFF
FIRMWARE	00080000	0008FFFF
CTRLNET	00090000	0009FFFF
LINUX	000A0000	000AFFFF
CIOS	000B0000	000BFFFF
MUDM	000C0000	000CFFFF
SPI	000D0000	000DFFFF
BGPM	000E0000	000EFFFF
TEST	FFFE0000	FFFEFFFF
RESERVED2	FFFF0000	FFFFFFF

7.1.2 Category

The category portion of the RAS message is the entity that encountered an error. Category names can be up to 16 characters long. Each category can be used by multiple components. Table 7-2 provides a list of possible RAS categories.

Table 7-2 RAS categories

RAS category	Description
BQC	Blue Gene/Q compute card
BGL	Blue Gene/Q link module
DDR	Double Data Rate Memory
PCI	PCI adapter card
Ethernet	Ethernet adapter card
InfiniBand	InfiniBand adapter card
AC_TO_DC_PWR	Bulk Power Supply
DC_TO_DC_PWR	Power Module
Cable	Cable
Message_Unit	Message unit
Card	Generic Card/Board
Clocks	Clocks
Clock_FPGA	Clock FPGA
Service_Card	Service Card
IO_Board	I/O Board
Node_Board	Node Board
Icon	Icon FPGA
Palomino	Palomino FPGA
DCA	Direct Current Assembly Card
Fan	Fan
Fan_Assembly	Fan Assembly
Optical_Module	Optical Module
Temp_Sensor	Temperature sensor on a card or board
Job	Job
Block	Block
Process	Process or daemon
Coolant_Monitor	Coolant Monitor
Software_Error	Software error condition
ELF_Image	ELF Image error condition

RAS category	Description
UPC	Universal Hardware Performance Counters

7.1.3 Severity

Each RAS message is assigned a severity. There are three possible severities of a RAS event:

FATAL	Designates a severe error event that presumably leads the application to fail or abort.
WARN	Designates potentially harmful situations, such as exceeding a soft error threshold or failure of a redundant component.
INFO	Designates informational messages that highlight the progress of system software.

7.1.4 Message

The message portion of the RAS event contains a brief overview of the reason for the event. Depending on the message, it might provide the location or type of error. The message can include substitution variables, such as a return code, torus coordinates, error count, and so forth. Therefore, the message text can vary for the same message ID.

7.1.5 Description

The description provides a more detailed message about the event. Typically in this area you see events that lead to an additional message, such as environmentals that are being read or that a specific diagnostic test is running. The description might also suggest the source of the problem. The message in Example 7-1 came from the description area of message ID 00061006:

Example 7-1 Message description from 00061006

The Health Check analyzed the coolant monitor at the specified location, and received an indication that one or more power faults occurred. This may be an indication that one of the bulk power enclosures on the midplane is not functioning properly.

7.1.6 Service action

A RAS event might include a recommendation for service action. Some errors might always result in the replacement of a compute node. If that is the case, the message indicates that it will resolve the problem. In other cases, you might be instructed to run diagnostics or to verify a reading on the Environmental Queries page to determine whether the event is a transient issue. Many of the recommendations to run diagnostics include the relevant test suites that you must select.

The action might specify a replacement policy based on the number of errors observed within a given time period. If the RAS event includes a *threshold count*, the service action will recommend that a part be replaced if the count of errors exceeds the threshold. The service action, replacement recommendations, and relevant diagnostic test buckets are shown in Example 7-2 on page 97.

Example 7-2 Sample RAS event

```

ID: 21093938
Time: 2011-09-29 08:12:44.550729
Message ID: 00040003
Category: Optical_Module
Component: MC
Severity: WARN
Block ID: DefaultControlEventListener
Location: R00-IE-009
Message: Optical module environmental data is unavailable: Avago=9, rc=0xFFFFFFFF/-2
Description: Unable to read the optical module's lower page. _avagos[xx]->read() failed.
Service Action: The part at this location has threshold count policy of 10 or more errors.
Run diagnostics. The relevant diagnostic bucket(s) to run are: nodeboard,ioboard.
Relevant Diagnostics nodeboard,ioboard
Threshold count: 10

```

7.1.7 Diagnostics suites

A list follows of relevant diagnostics suites to execute to better isolate the condition of a RAS event:

checkup	A series of tests designed to provide maximum coverage for a typical 30 minutes run time for a midplane-sized block.
small	A quick diagnostics run providing less fault detection.
medium	A medium length diagnostics run providing moderate fault detection.
large	A long length diagnostics run providing high fault detection.
complete	Run all diagnostics, taking the longest time and providing the highest fault detection.
servicecard	All service card specific tests.
nodeboard	All node board specific tests.
ioboard	All I/O board specific tests.
memory	Tests that target the memory subsystem on the nodes.
processor	Tests that target the BQC processors.
torus	Tests that target the torus network.
power	Tests that target the power subsystem.
pcie	Tests that target the PCIe hardware.

7.1.8 Control Action

The Control Action identifies an action to be taken by the Control System when a RAS event occurs. Multiple Control Actions can be specified.

Certain RAS events might specify Control Actions to end the job and free the block. Although the event is fatal to the job, the condition might be transient or soft, which means that hardware failures occur and cause an application to fail, but that the application can be restarted on the same hardware without hardware repair. Soft errors are most likely radiation induced and are a natural consequence of the impact of terrestrial and cosmic radiation sources, as well as radioactive impurities in the semiconductor and packaging materials close to the CMOS circuits. Soft errors can also occur because of defects in system software.

Other failures are hard. Hard failures mean that the failed part permanently exhibits the failure mode, and the hardware must be replaced to fix the failure.

A RAS event can include multiple Control Actions that indicate that the condition is likely hard and warrants marking the hardware in error. Marking hardware in error helps job schedulers avoid dispatching jobs to failing hardware. The possible Control Actions are:

COMPUTE_IN_ERROR	This action marks a compute node or I/O node in Error state in the database. Any compute block that uses the compute node is prevented from booting. Any I/O block that uses the I/O node will boot, but any I/O links from compute nodes to the failed I/O node are not usable.
BOARD_IN_ERROR	This action marks a node board or I/O drawer in Error state in the database and prevents any blocks that use the node board or I/O drawer from booting.
CABLE_IN_ERROR	This action marks a cable in Error state in the database and prevents any blocks that use the cable from booting.
END_JOB	This action ends all jobs associated with the compute node, I/O node, midplane, or node board.
FREE_COMPUTE_BLOCK	This action frees a block associated with the compute node.
SOFTWARE_IN_ERROR	This action marks a node in software error state in the database. The state is reset back to the Available state upon freeing the block.
BQL_SPARE	This action spares the bad wire on the link chip.
RACK_IN_ERROR	This action marks all node boards and I/O drawers in the rack in Error state in the database and prevents any blocks that use the node boards and I/O drawers in the rack from booting.
DCA_IN_ERROR	This action marks the DCA in Error state in the database.

7.2 Tailoring RAS messages

The Control System has a mechanism to alter RAS events. These alterations are typically guided by IBM Support as needed to circumvent a problem while waiting for a bug fix. The mechanism reads a set of change specifications from the `ras_environment_filter.xml` file in the `/bgsys/local/etc` directory during server start. If the file is missing from that location, the mechanism reads the file from its original installation location at `/bgsys/drivers/ppcfloor/ras/etc`.

The change specifications indicate the message ID of interest and lists the RAS event item to change based on its key name. Example 7-3 shows how to alter the outcome of a RAS event.

Example 7-3 Sample `ras_environment_filter.xml` file

```

<BgRasEnvironments>
  <BgRasEnvironment environment="PROD">
    <!-- Change the severity of an event -->
    <RasEventChangeSpec id="xxxxxxx" key="BG_SEV" value="WARN"/>
    <!-- Change the control action - this action marks the compute node in Error -->
    <RasEventChangeSpec id="xxxxxxx" key="BG_CTL_ACT" value="COMPUTE_IN_ERROR"/>
    <!-- Change the control action to not mark the node in Error -->
    <RasEventChangeSpec id="xxxxxxx" key="BG_CTL_ACT" value="NONE"/>
  </BgRasEnvironment>
</BgRasEnvironments>

```

The first occurrence of RasEventChangeSpec shows how to change a RAS event severity. The second RasEventChangeSpec shows how to add a Control Action to mark the hardware in error. Marking the hardware that is in error makes the hardware unavailable until a service action is performed. The third RasEventChangeSpec shows how to prevent the hardware from being marked in error. The Control Action is changed to NONE.

7.3 Viewing Blue Gene/Q defined RAS events

Blue Gene/Q defined RAS events can be viewed through Blue Gene Navigator by clicking the RAS book link under the Knowledge Center tab (see Chapter 2.2.15, “Knowledge Center” on page 27).

This information can also be found in the `/bgsys/drivers/ppcfloor/ras/etc/RasEventBook.htm` file.

7.4 Viewing sent RAS events

RAS events sent by the system or users are stored in a database and can be viewed through Blue Gene Navigator by clicking the RAS tab (see section 2.2.8, “RAS” on page 17).



Toolkit for Event Analysis and Logging

The Toolkit for Event Analysis and Logging (TEAL) is an IBM high-performance computing (HPC) event analysis framework that is based on IBM Blue Gene/P® Event Log Analysis (ELA) and Federation ELA. It is designed as a pluggable processing pipeline that allows different components to use connectors to log events, analyze the events, create and log alerts, and deliver the alerts to interested parties.

TEAL is available as open source on sourceforge to allow the HPC community to enhance the toolkit and expand it to more platforms. This chapter describes the Blue Gene/Q-specific information. More information about base TEAL framework is at:

<http://sourceforge.net/apps/mediawiki/pyteal/index.php>

8.1 TEAL framework

TEAL provides a central location for logging and analyzing events. Its processing is designed as a pipeline that allows analysis and delivery plug-ins to be used. TEAL supports the processing of events as they occur (real-time) and events that occurred in the past (historic).

The TEAL processing pipeline consists of analyzers, filters, and listeners. The framework does not provide any specific analyzers; instead, it provides the ability to plug in component-specific and cross-component (system) analyzers. Analyzers are written to TEAL's analyzer plug-in interface. More information can be found at:

http://sourceforge.net/apps/mediawiki/pyteal/index.php?title=HowTo_-_Event_Analyzer

TEAL supports both component-specific analysis and cross-component analysis. Each component requires a connector to the TEAL framework, which takes events from a specific component and translates them into a common event structure.

TEAL framework provides some configurable filters and listeners that provide a flexible alert delivery system. Listeners are configured with a list of filters that an alert must pass before it is given to them. Filters analyze an alert and determine if the alert is sent to its associated listener. Figure 8-1 shows the TEAL structure and the communication flow in the pipeline.

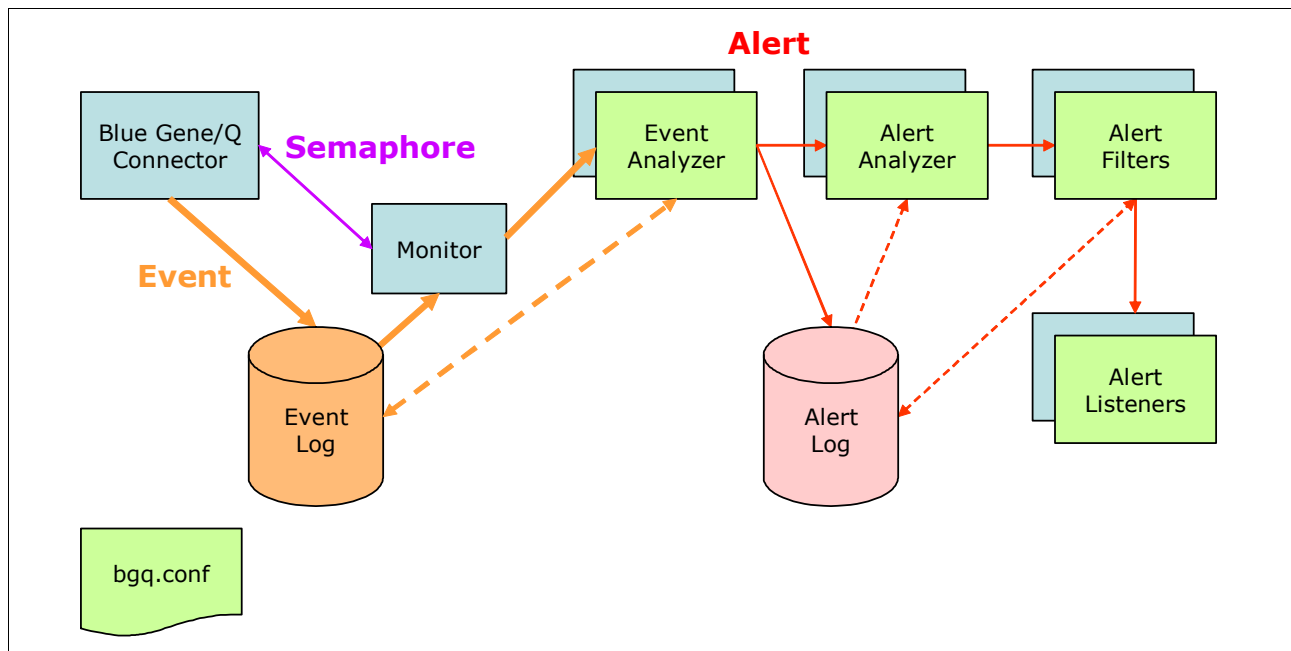


Figure 8-1 TEAL structure

The components of TEAL are:

Connector	Connectors are external to the TEAL process and are responsible for taking events in the component, translating the event(s) and notifying TEAL that new events were added.
Monitor	A monitor is responsible for detecting new event insertions and instantiating and putting in the processing pipeline for analysis.
Event Analyzer	Analyzes events for a specific component.
Alert Analyzer	Analyzes alerts from multiple components.

Alert Filter	The Alert Filter is part of the alert reporting pipeline that is responsible for determining if a particular Alert Listener will receive notification of the alert.
Alert Listener	Alert Listeners are also part of the alert reporting pipeline, responsible for taking an alert and reporting to an external medium.
Event Log	Event Logs are a common repository for all events from all components within the system.
Alert Log	An Alert Log is a single repository for all alerts that occur within a system. All alerts, whether reported or not, are contained in this log.

The TEAL configuration file, `bgq.conf`, is located in the `/etc/teal` directory. It is used to define and enable any TEAL plug-ins. The configuration file is shipped with default settings that can be modified as needed for a specific environment.

8.1.1 Connector

Blue Gene/Q provides a connector to the TEAL framework. The connector is responsible for getting Blue Gene/Q RAS events, converting them into common TEAL event log information, and notifying TEAL of new events.

The connector uses the Real-time API to register for RAS events so that it will be notified by the Real-time server when a RAS event of interest is logged. In the case where the Real-time server is not available, the connector switches to periodic mode automatically and retrieves RAS events within a specified polling interval until the Real-time server is available again.

8.1.2 Event analyzers

Blue Gene/Q provides multiple implementations of event analyzers that analyze RAS events and send alerts when necessary. Here are the types of event analyzers provided:

HardwareInError	Analyzes events that cause hardware to be marked in error. This event analyzer covers RAS events that have the following Control Actions: <code>COMPUTE_IN_ERROR</code> , <code>BOARD_IN_ERROR</code> , <code>CABLE_IN_ERROR</code> , <code>RACK_IN_ERROR</code> , <code>DCA_IN_ERROR</code> .
JobFatal	Analyzes events that cause a job to end (RAS events with <code>END_JOB</code> Control Action).
ThresholdExceeded	Analyzes events if the threshold is reached or exceeded (RAS events with non-NULL threshold count and threshold period).
BqlEventAnalyzer	Analyzes BQL RAS events and determines whether the BQL threshold of errors has been reached or exceeded.
CommonMode	Analyzes events for a common hardware location. For example, if two different compute nodes (<code>R00-M0-N0-J00</code> and <code>R00-M0-N0-J01</code>) report the same problem, the underlying cause might be with the node board <code>R00-M0-N0</code> (the hardware location they have in common).

8.1.3 Plug-ins

The following plug-ins are supported by Blue Gene/Q:

Monitors

Real-time	TEAL framework is running and processing events as they occur.
Historic	TEAL framework is started discretely and processing past events based on the criteria supplied by the user.

Alert filters

Duplicate	Causes TEAL to stop reporting alerts to a listener that have the same alert ID and location and equal or lower severity and priority.
Noise	Stops reporting alerts to a listener that match various alert fields with regular expression.

Alert listeners

Smtip	Sends email to a set of users when an alert is generated.
File	Writes alerts to a file or stdout / stderr in a specified format. Mainly used for historical analysis.
Call	Calls an external program to handle an alert.

8.2 Location reporting

Location, which points to a specific event location, can be physical or logical. It is a key concept used within TEAL framework for scoping and pin-pointing errors. Location is defined in a hierarchical structure, where there might be one or more types per level. It can be scoped to the various levels to see the grouping of events.

The TEAL location syntax for Blue Gene/Q is structured as follows:

Application	A:<node>##<application>##<pid>
Job	J:<job-id>##<block-id>
Compute	C:<compute-location>
IO	I:<I/O-location>

8.3 Integration with BGmaster

The base TEAL framework and Blue Gene/Q connector can be configured to start and stop through BGmaster in the `bg.properties` file (default is in `/bgsys/local/etc/bg.properties`) as shown in Example 8-1. In the `[master.binmap]` section set the path and name of the TEAL binary used to start the servers. The entries in the `[master.binargs]` section list the arguments to be passed to the TEAL binary. The arguments can be modified as needed, for example, if debug messages are needed, the `"-m debug"` option can be added to the arguments. The `[master.startup]` section specifies the order the various binaries should be started in.

By default the TEAL log files are located in the `/bgsys/logs/BGQ/teal` directory.

Example 8-1 BGmaster sections of bg.properties

```
[master.binmap]
```

```

# Maps an alias to a binary path.
teal_server = /opt/teal/ibm/teal/teal.py
teal_bg = /opt/teal/ibm/bgq/bgq.py

[master.logdirs]
# Maps aliases to logging directories. By default, all binaries will log to
# logdir in the [master.server] section.
teal_bg = /bgsys/logs/BGQ/teal/
teal_server = /bgsys/logs/BGQ/teal/

[master.binargs]
# List of arguments for each alias specified in the [master.binmap] section.
teal_server = --realtime --logfile /dev/stdout
teal_bg = --logfile /dev/stdout

[master.startup]
start_order=realtime_server,mmcs_server,bgws_server,runjob_server,runjob_mux,mc_server,teal
_server,teal_bg

```

8.4 Alerts and service actions

Alerts created by TEAL are closed following a successful completion of a service action. For information about service actions, see Chapter 9. “Service actions” on page 107.

There are two choices on how to close alerts:

- | | |
|--------------|------------------------------------------------------------------------------------------------------------------|
| False | Set to <i>false</i> to close alerts only on those replaced devices on or in the serviced hardware. |
| True | Set to <i>true</i> to close alerts on all devices on or in the serviced hardware even if they were not replaced. |

For example, if **ServiceNodeBoard** was used to replace a compute card, and if `CloseAllAlertsOnServiceNodeBoard` was set to *false*, only those alerts on the replaced compute card are closed. If it was set to *true*, alerts on the node board and all the devices on the board (for example, compute card, link chips, optical module, node DCA) are closed, even if they were not replaced.

The configuration settings for how to close alerts is in the `[baremetal]` section of the `bg.properties` file, as shown in Example 8-2.

Example 8-2 [baremetal] section of bg.properties

```

[baremetal]
CloseAllAlertsOnServiceBulkPowerModule = false
CloseAllAlertsOnServiceClockCard = false
CloseAllAlertsOnServiceIoDrawer = false
CloseAllAlertsOnServiceMidplane = false
CloseAllAlertsOnServiceNodeBoard = false
CloseAllAlertsOnServiceNodeDCA = false
CloseAllAlertsOnServiceRack = false

```

8.5 Installation

The following prerequisite software is required prior to installing TEAL:

- ▶ Python 2.6 is provided with RHEL 6.
- ▶ Pyodbc, which is available at:

http://sourceforge.net/apps/mediawiki/pyteal/index.php?title=Building_pyodbc

There are two RPMs required for TEAL installation:

- ▶ Base TEAL framework which is provided with the Blue Gene/Q driver
- ▶ Blue Gene/Q Connector which is provided with the Blue Gene/Q driver

Figure 8-2 shows the directory structure after TEAL is installed.

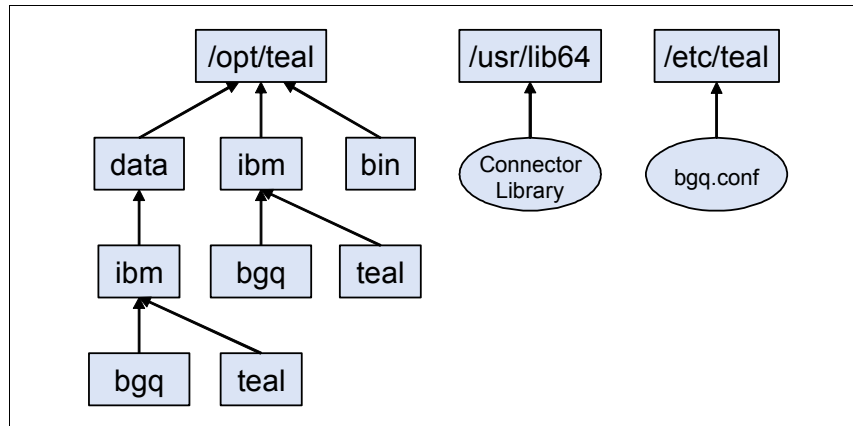


Figure 8-2 TEAL directory structure



9

Service actions

Preparing the Blue Gene/Q hardware for service is a normal part of the System Administrator's duties. This chapter describes service actions and discusses how to prepare the hardware for service and how to cycle power.

9.1 Service actions overview

Replacing hardware on the Blue Gene/Q system requires you to perform a service action. Service actions prepare the hardware for removal by a trained IBM Systems Service Representative (SSR). You can implement service actions using either the Blue Gene Navigator (discussed in Chapter 2., “Navigator” on page 7) or the command line on the service node.

The following hardware can be prepared for service with the service action commands:

- ▶ Node board, including any compute cards on a node board
- ▶ Node Direct Current Assembly (DCA) on a node board
- ▶ Service card
- ▶ I/O drawers, including any compute cards, PCI adapter cards and fans in an I/O drawer
- ▶ Bulk Power Module (BPM) in a Bulk Power Enclosure (BPE)
- ▶ Clock card
- ▶ Any hardware that requires tools to service (e.g., BPE, midplane, entire rack)

As a general rule, if you can remove the hardware with your fingers, you can generate a service action for that location prior to removing the part. If, however, you need to use tools to perform the service, you must turn off the power to the rack before removing it.

In most cases, at least one BPM must be powered on in each BPE at all times during a service action so that the persistent power is maintained to the clock cards and the coolant monitor in the rack. More details are discussed in section 9.3, “Hardware maintenance” on page 110.

9.2 Service action commands

There are nine service action commands: two for the system installation or upgrade and seven for hardware maintenance. Service action commands are located in the `/bgsys/drivers/ppcfloor/baremetal/bin` directory. This section provides a description and syntax for these commands.

In addition to the unique options, all commands have the options specified in the following list:

--user <userId>

The user name to be specified when connecting to MC server. If not specified the user name defaults to the user ID invoking this command.

--msglevel <level>

Controls the additional output level. The possible values are:

- **VERBOSE**: Provides detailed output messages.
- **DEBUG**: Provides more extensive output messages useful for debugging.

--log <path>

The path to the directory that contains the log file generated from the command. The default is `/bgsys/logs/BGQ/ServiceAction`.

--properties <filename>

The fully qualified file name to the Blue Gene/Q configuration file. The default file, /bgsys/local/etc/bg.properties or the file name specified by BG_PROPERTIES_FILE environment variable if defined is used if the option is not specified.

--help

Displays the help text for the command.

9.2.1 InstallServiceAction

The **InstallServiceAction** command is used to discover all hardware components in the system and to populate the database with the current state of those hardware components when the Control System is started. Use the **InstallServiceAction** command only when the hardware is brought up for the first time or when the system is upgraded with additional racks after the Control System is restarted. The syntax of the command is:

InstallServiceAction LocationString [OPTIONS ...]

The accepted arguments for the **InstallServiceAction** command are:

LocationString

The rack location string to update the database with information about the current state of the hardware (active, missing, or in error). Supported location strings include:

ALL	All compute and I/O racks in the machine. This is the default and must be specified for the first time installation.
Rxx:Ryy	All compute and I/O racks in the specified range of rows and columns. For example, R50:R5V, for all compute and I/O racks in row five.
Rxx or Qxx	Specific compute or I/O rack.

--dbsummary

Display the database summary of all the hardware in the specified racks.

When the system is installed for the first time, **ALL** (the default) must be specified even if not all of the racks are initially installed so that the accurate rack configuration is reflected in the database. Whenever additional racks are installed as the system is upgraded, the Control System must be restarted to allow **InstallServiceAction** to discover newly added hardware components.

9.2.2 VerifyCables

The **VerifyCables** command is used to verify the data cable connections between a set of node boards and I/O drawers including the connections between the link chips, optical modules, and the data cable ports. The syntax of the command is:

VerifyCables LocationString [OPTIONS ...]

The supported arguments for the **VerifyCables** command are:

LocationString

The hardware location string to update the database with information about the current state of the cables. Supported location strings include:

ALL	All node boards and I/O drawers in the system. This option must be specified for the first time installation.
Rxx:Ryy	All node boards and I/O drawers in the specified range of racks (for example, R50:R5V for all compute and I/O racks in row five).
Rxx or Qxx	All node boards and I/O drawers in the specified rack.
Rxx-Mx-N	All node boards in the same midplane.
Rxx-Mx-Nxx	Specific node board.
Rxx-lx or Qxx-lx	Specific I/O drawer.

--dbsummary

Displays the database summary of all the data cables at the specified location.

When the system is installed for the first time, ALL must be specified even if not all the racks are initially installed so that the accurate cable configuration is reflected in the database.

9.3 Hardware maintenance

There are seven service action commands that can be used when hardware replacement is necessary:

- ▶ **ServiceNodeDCA**
- ▶ **ServiceNodeBoard**
- ▶ **ServiceIoDrawer**
- ▶ **ServiceMidplane**
- ▶ **ServiceBulkPowerModule**
- ▶ **ServiceRack**
- ▶ **ServiceClockCard**

The syntax of these commands is:

ServiceXXXXX LocationString Action [OPTIONS ...]

ServiceXXXXX refers to a specific service action command. The supported arguments for the service action command are:

LocationString

The location string for the hardware to service. Supported location strings include:

ServiceNodeDCA	Rxx-Mx-Nxx-Dx - Node DCA
ServiceNodeBoard	Rxx-Mx-Nxx - Node board Rxx-Mx-N - All node boards in a midplane
ServiceIoDrawer	Rxx-lx - I/O drawer in a compute rack Qxx-lx - I/O drawer in an I/O rack
ServiceMidplane	Rxx-Mx - Service card, all node boards, all BPMs, and all I/O drawers associated with the midplane (see Table 9-1 on page 114 for a list of BPMs and I/O drawers associated with each midplane)
ServiceBulkPowerModule	Rxx-Bx-Px - BPM in a compute rack Qxx-B-Px - BPM in an I/O rack

ServiceRack	Rxx - Specific compute rack Qxx - Specific I/O rack
ServiceClockCard	Rxx-K - Clock card in a compute rack Qxx-K - Clock card in an I/O rack

Action

Supported action strings are:

PREPARE	Prepares the specified hardware for service.
END	Ends the service action on the specified hardware.
CLOSE	Forces an existing open or failed service action to the closed state so that the service action can be restarted.

In general, if a service action fails, the status of the service action is marked in error, and all affected hardware components are marked in either service or error in the database depending on the reason for the failure. After the error is corrected, the service action must be closed and prepared again.

9.3.1 ServiceNodeDCA

The **ServiceNodeDCA** command is used when only one node DCA on a node board needs to be replaced. When a service action is prepared for a node DCA, the power regulators on the node DCA are powered off so that the DCA can be removed safely from the node board. When the service action is ended, the command powers the node DCA back on and verifies that the replaced node DCA is fully functional.

9.3.2 ServiceNodeBoard

The **ServiceNodeBoard** command is used when any compute cards on a node board, both DCAs on a node board, or an entire node board need to be replaced. When a service action is prepared for a node board, the power regulators on the node board and DCAs are powered off so that the power cables can be safely removed from the DCAs. When the service action is ended, the command powers on the node board and DCAs, initializes the node board, and verifies that the node board, all compute cards, both DCAs on the node board, and all data cables connected to the node board, are fully functional.

9.3.3 ServiceIoDrawer

The **ServiceIoDrawer** command is used when any compute cards, PCIe adapters, or fan modules in an I/O drawer, or an entire I/O drawer, need to be replaced. When a service action is prepared for an I/O drawer, all power regulators in the I/O drawer are powered off so that the power cable can be safely removed from the I/O drawer. When the service action is ended, the command powers on and initializes the I/O drawer and verifies that the I/O drawer, all compute cards on the I/O drawer, and all data cables connected to the I/O drawer are fully functional.

9.3.4 ServiceMidplane

The **ServiceMidplane** command is used when the service card, any node boards in the midplane, or any BPMs or I/O drawers associated with the midplane need to be replaced. For example, midplane R00-M1 includes R00-M1-S, all the node boards in R00-M1, I/O drawers

R00-IC and R00-IF, and all BPMs in BPEs R00-B2 and R00-B3. Refer to Table 9-1 on page 114 for a list of BPMs and I/O drawers associated with each midplane.

When a service action is prepared for a midplane, all power regulators on the service card, all node boards in the midplane, and I/O drawers associated with the midplane are powered off so that these devices can be safely removed. When the service action is ended, the command initializes the service card, verifies that required number of BPMs controlled by the service card are fully functional, then powers on and initializes the node boards and I/O drawers associated with the midplane, and finally verifies that these devices are fully functional.

Because the service card gets initialized by the **ServiceMidplane** command, there are some factors that must be considered when using this command:

- ▶ If the service card controls the master clock card, initializing the service card also resets the master clock card affecting all the active jobs in the system.
- ▶ The environmental data (for example, voltage, temperature) on all node boards, BPMs, and the coolant monitor that are controlled by the service card will not be available while the service card is being serviced.

In addition, at least one BPM in each BPE must be left powered on at all times during the service action so that the persistent power is maintained to the clock cards and coolant monitor.

Therefore, consider using other service action commands instead of the **ServiceMidplane** command unless the service card needs to be replaced. Refer to Table 9-2 on page 114 when deciding which service action command to use. For example, if several node boards in the same midplane must be replaced at the same time, use **ServiceNodeBoard** instead of **ServiceMidplane**.

9.3.5 ServiceBulkPowerModule

The **ServiceBulkPowerModule** command is used to service a BPM in a compute or I/O rack. Because of the N+1 redundancy supported by each BPE, one BPM per BPE can be replaced without affecting the system operation. When a service action is prepared for a BPM, the command verifies that there are no other BPMs in the same BPE that are being serviced and that the remaining BPMs are operational unless all the node boards and I/O drawers associated with the same midplane (or all I/O drawers in the I/O rack) are also being serviced. When the service action is ended, the command verifies that the BPM is fully functional.

9.3.6 ServiceRack

The **ServiceRack** command is used when any combination of the following hardware components in a compute rack or an I/O rack need to be replaced:

- ▶ Service card that does not control the master clock card.
- ▶ Any node boards, including any compute cards and node DCAs on a node board.
- ▶ Any I/O drawers, including any compute cards, PCIe adapters, and fan modules in an I/O drawer.
- ▶ Any BPMs as long as at least one BPM can be left powered on to maintain the persistent power to the clock cards. In a compute rack, leave at least one BPM in BPE B1 or B3 powered on.

When a service action is prepared for a rack, the power regulators on all node boards and I/O drawers are powered off. In addition, you must turn off the power as follows:

1. Use the BPE kill switch to turn off the power to all node boards and I/O drawers in the rack. For a compute rack, turn off the BPE kill switch on two BPEs on either the front or back side of the rack because the kill switches on two BPEs for each midplane are linked together.
2. For a compute rack only, turn off the AC circuit breaker for all BPMs in BPEs B0 to B2 to disconnect the persistent power to the service cards.

When the service action is ended, the command brings back both midplanes into a fully functional state in the same way described for the **ServiceMidplane** command.

9.3.7 ServiceClockCard

The **ServiceClockCard** command is used when any combination of the following hardware components in a compute rack or an I/O rack need to be replaced:

- ▶ Any clock cards
- ▶ Any service cards
- ▶ Any node boards, including any compute cards and node DCAs on a node board
- ▶ Any I/O drawers, including any compute cards, PCIe adapters, and fan modules in an I/O drawer
- ▶ Any BPMs
- ▶ Any hardware that require tools to service (for example, BPE, midplane, coolant monitor, or entire rack)

When a service action is prepared, all node boards and I/O drawers in the rack are powered off, and all the jobs running in the midplanes and I/O drawers that are connected to the clock cards in the rack are terminated. In addition, if there are any downstream racks in the clock topology, all node boards in I/O drawers in these racks are powered off and the jobs running in these racks are terminated. The user must turn off the AC power to all BPEs in the rack then terminate power to the rack at the building power circuit breakers. When the service action is ended, the command returns both midplanes and then the rest of the downstream racks into a fully functional state in the same way described for the **ServiceMidplane** command.

9.4 Deciding which service action command to use

Deciding which service action command to use while minimizing the impact on the jobs that are running in the system is affected by two factors: (1) how the power is distributed in the rack, and (2) which service card controls the master clock card. A compute rack contains two midplanes, and each midplane contains two BPEs with 9 BPMs each. A BPE provides the power to 8 node boards and up to one I/O drawer. There is one service card in each midplane, and a service card is powered by one BPE. The coolant monitor is powered by two BPEs, and the clock card is powered by two other BPEs. There is one clock card in each compute rack, and the master clock card is controlled by one service card that can be in either midplane. An I/O rack contains one BPE with 6 BPMs, and a BPE provides the power to up to 12 I/O drawers and up to two clock cards in the rack. The master I/O drawer at position I5 (for example, Q01-I5) controls all BPMs in the rack. Table 9-1 on page 114 shows how the power is distributed to various hardware components in a compute and I/O rack.

Table 9-1 Power distribution

Midplane	BPE	BPM	Service card	Node boards	I/O drawers	Coolant monitor	Clock card
Rxx-M0	Rxx-B0	Rxx-B0-P0 to P8	Rxx-M0-S	Rxx-M0-N00 to N07	Rxx-IE	Rxx-L	n/a
	Rxx-B1	Rxx-B1-P0 to P8	n/a	Rxx-M0-N08 to N15	Rxx-ID ^a	n/a	Rxx-K
Rxx-M1	Rxx-B2	Rxx-B2-P0 to P8	Rxx-M1-S	Rxx-M1-N00 to N07	Rxx-IC	Rxx-L	n/a
	Rxx-B3	Rxx-B3-P0 to P8	n/a	Rxx-M0-N08 to N15	Rxx-IF	n/a	Rxx-K
n/a	Qxx-B	Qxx-B-P0 to P5	n/a	n/a	Qxx-IO to IB	n/a	Qxx-Kx

a. = Standard location for the I/O drawer in a compute rack configured with one I/O drawer

Important: Unless the clock cards are being serviced, at least one functioning BPM in each BPE must be left powered on at all times during a service action so that the persistent power is maintained to the clock cards and the coolant monitor.

Use Table 9-2 as a guide when deciding which service action commands to use. Note that the master service card refers to the service card that controls the master clock card (can be either Rxx-M0-S or Rxx-M1-S), and the master midplane refers to the midplane that contains the master service card.

Table 9-2 Service action selection

Hardware to service	Service action command to use
One node DCA	ServiceNodeDCA Rxx-Mx-Nxx-Dx
One node board, any compute cards or both DCAs on the node board	ServiceNodeBoard Rxx-Mx-Nxx
Any or all node boards in a midplane	ServiceNodeBoard Rxx-Mx-N
One I/O drawer, any compute cards, PCIe adapters, or fan modules in the I/O drawer	ServiceIoDrawer Rxx-Ix or Qxx-Ix
Any or all I/O drawers in an I/O rack	ServiceRack Qxx
Master service card	ServiceClockCard Rxx-K
Non-master service card	ServiceMidplane Rxx-Mx
One BPM per BPE in a compute or I/O rack	ServiceBulkPowerModule Rxx-Bx-Px
All BPMs in BPE B0 or B2 (at the same time) in the master midplane NOTE: The BPE switch must be toggled to the off position for all four BPEs then power off the rack at the building power circuit breakers.	ServiceClockCard Rxx-K

Hardware to service	Service action command to use
2 to 6 BPMs (at the same time) in the master midplane	Use the following steps instead of using ServiceMidplane because ServiceMidplane on the master midplane results in resetting the master clock card. <ol style="list-style-type: none"> 1. ServiceNodeBoard Rxx-Mx-N (resets all the node boards in the midplane) 2. ServiceIoDrawer Rxx-Ix (for each I/O drawer associated with the midplane) 3. ServiceBulkPowerModule Rxx-Mx-Bx-Px (for each BPM to service) Must leave at least three BPMs per BPE powered on for the service card, clock card, and coolant monitor.
2 to 6 BPMs (at the same time) in the non-master midplanes	ServiceMidplane Rxx-Mx Must leave at least three BPMs per BPE powered on for the service card, clock card, and coolant monitor.
2 to 5 BPMs (at the same time) in an I/O rack	ServiceRack Qxx Must leave at least one BPM powered on for the clock cards.
All BPMs (at the same time) in an I/O rack	ServiceClockCard Qxx-K
Any I/O drawers or BPMs in an I/O rack	ServiceRack Qxx Must leave at least one BPM powered on for the clock cards.
Any BPMs, service cards, node boards, I/O drawers in a compute rack that contains the master clock card	ServiceClockCard Rxx-K
Any BPMs, service cards, node boards, I/O drawers in a compute rack that does not contain the master clock card	ServiceRack Rxx Must leave at least one BPM in BPE B1 or B3 powered on for the clock card.
Clock card	ServiceClockCard Rxx-K or Qxx-K
Hardware that requires tools to remove and replace (for example, BPE, coolant monitor, midplane, or rack)	ServiceClockCard Rxx-K or Qxx-K

9.5 Preparing a service action

In general, when you initiate a service action, the service action command performs the following checks and internal procedures:

- ▶ A check is made to ensure that there are no conflicting service actions in progress. A software imposed limitation of only one open service action is allowed per hardware component. For example, if a node board is already being serviced, a request to service the same node board or the midplane that contains the node board will result in an error.
- ▶ The status of all affected hardware components are changed to the service status in the database. Job schedulers will not allocate hardware resources that have service status. Refer to Table 9-3 on page 116 for a list of affected hardware components for each service action command.

- ▶ Jobs that are using affected hardware are ended, and blocks are freed. For the **ServiceClockCard** command, the jobs running in the downstream midplanes and I/O drawers in the clock topology are also ended.
- ▶ A reliability, availability, and serviceability (RAS) event is written to the event logs stating that a service action has started on a specific hardware along with the name of the person who initiated the action.
- ▶ A request is made to MC server to open a target set for the affected hardware. If another target set that includes the same hardware is already opened by another job, the request fails; consequently, the command retries the request again for several minutes. For example, the Health Check that periodically collects the environmental data from all hardware components might cause a target set conflict for a few seconds forcing the service action command to retry the request. If the request still fails after several retries, the service action fails with an error leaving all affected hardware in the service status. The service action must be prepared again after the conflict is resolved.
- ▶ The RAS analysis is performed on the hardware, and the Vital Product Data (VPD) is updated in the database and on the hardware.
- ▶ The affected hardware components are powered off as necessary so that they can be safely replaced.
- ▶ A RAS event is posted indicating that the hardware is ready for service.

Table 9-3 Service action commands and affected hardware components

Command	Service card	Node board	Node DCA	Node	I/O drawer	I/O node	Data cable	BPM	Clock card
ServiceNodeDCA			X						
ServiceNodeBoard		X	X	X			X		
ServiceIoDrawer					X	X	X		
ServiceMidplane	X	X	X	X	X	X	X	X	
ServiceBulkPowerModule								X	
ServiceRack (compute rack)	X	X	X	X	X	X	X	X	
ServiceRack (I/O rack)					X	X	X	X	
ServiceClockCard	X	X	X	X	X	X	X	X	X
ServiceClockCard (I/O rack)					X	X	X	X	X

9.6 Ending a service action

After the hardware is replaced, the hardware can be brought back to a fully functional state by ending the service action with the appropriate service action command. In general, the service action command performs the following internal procedures to end the service action:

1. A check is made to validate that the requested service action is still open.
2. The target set is opened for the affected hardware.
3. All affected hardware components are initialized.

4. The status of affected hardware is changed to the missing status in the database to indicate that they are not being serviced any longer and are not discovered yet.
5. All the affected hardware components are rediscovered.
6. The replaced hardware components are verified to ensure that they are fully functional by performing a set of diagnostic tests.
7. If node boards or I/O drawers with data cables are included in the service action, those cables are verified. If the cable verification failed or could not be performed, and some cables resulted in the non-active status, the service action will still complete but the cable verification must be performed again with the **VerifyCables** command after the error conditions are corrected.
8. The status of the hardware is changed to the active status in the database.
9. Job scheduling is allowed on the hardware.
10. The status of all affected hardware is verified in the database.
11. A RAS event is written to the event logs stating that a service action has ended on a specific hardware.

9.7 Service action logs

The service action commands document the steps taken to prepare or end the requested service action along with any informational, warning, or failure messages chronologically in the service action log in the `/bgsys/logs/BGQ/ServiceAction` directory. The logs can be viewed from the command line. In general, a warning message is identified with an at sign (`@`) and a failure message with an asterisk (`!`). The log file name uses the following syntax:

```
<service action command>-<location>-<timestamp>.log.
```

For example, a service action started for node board R01-M0-N04 at 13:32:25 on 7/6/2011 is named: `ServiceNodeBoard-R01-M0-N04-2011-0706-13:32:25.log`. When that service action is ended, another log with a similar name is created but the time stamp will be different. If the service action from the previous example completes after 11 minutes, the log file name is: `ServiceNodeBoard-R01-M0-N04-2011-0706-13:43:25.log`.

9.8 Cycling power

The process to cycle power on your system depends on the reason you are shutting down the system and also on the number of Blue Gene/Q racks in your system. Any time you replace parts within an individual rack, you must run a service action to prepare the hardware for service. When the service action ends, the database is updated to reflect the new hardware that was installed. If you are shutting down the system with no intention of replacing hardware you can use an alternate process that is described in this section. Keep in mind that, in a multiple rack system, turning off a rack with a controlling clock in it (master clock, secondary,) affects all of the racks that are downstream in the clock network. Use the following steps to cycle power on the system when no hardware will be replaced:

1. Stop BGmaster using the following command (under the `bgqadmin` profile):

```
$ master_stop bgmaster
```
2. Set the BPE switch to the *off* position on all four BPEs.

3. Power off the rack by terminating power at the building power circuit breakers. Use the facility breakers because there is still persistent power present when the power switches on the BPEs and BPMs are in the off position.

Use the following steps to power the system back on:

1. Turn the facility breakers back on.
2. Set all of the BPE switches to the *on* position.
3. Restart BGmaster with the following command:

```
$ master_start bgmaster
```




Diagnostics

This chapter describes the diagnostics suite that is provided with Blue Gene/Q and demonstrates how to run the Blue Gene/Q diagnostic suite from the command line.

10.1 System diagnostics

The Blue Gene/Q diagnostic suite interacts with the MMCS server to run tests on the Blue Gene/Q 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 systemic.

Diagnostics must be executed from `/bgsys/driver/ppcfloor/diags/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 section 2.2.12, “Diagnostics” on page 21.

Diagnostics include tests for:

- ▶ Memory subsystem
- ▶ Compute logic
- ▶ Instruction units
- ▶ Floating point units
- ▶ Torus network
- ▶ Internal and external chip communication
- ▶ Electrical and optical links
- ▶ PCIe bus and interface

10.1.1 Requirements

To control the hardware, the MMCS server and `mc_server` must be running.

Users who execute diagnostics must have read and execute authority to the `/bgsys/drivers/ppcfloor/diags` directory and subdirectories. The output that the diagnostics generate is written to the `/bgsys/logs/BGQ/diags` directory. Each time you run the diagnostics, a new directory is created in the `/bgsys/logs/BGQ/diags` directory. These new directories use 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 is a subdirectory 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, the `output.log` and `rasFile.log` files are written to the test case subdirectory. The user that initiates the diagnostic run must have all authority to each of these log directories. If the test case completes successfully, the diagnostics logs are deleted to save disk space. Service nodes are set up by default to automatically clean up any diagnostics log older than 30 days.

10.2 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 have an issue that requires that you run a specific type of test, 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 the diagnostics suite from the Navigator, you can base the run on the amount of time that you allowed.

Note: Run these tests with care. For example, the bqcbist, bqchssbist, bqlbist, and bqlhssbist tests are intensive when it comes to service node resources and can prevent the Control System from booting blocks in a timely manner. For best results, it is recommended that you do not run these tests when attempting to boot other blocks.

Certain diagnostics, such as bqlbist and bqlhssbist, require exclusive access to the link chips. If you are running them, be sure that other blocks are kept from using the affected link chips. These tests are not included in any of the buckets to avoid causing problems for link chips that are used for pass through.

The following test buckets are available on the command line:

checkup	A series of tests that are designed to provide maximum coverage for approximately 30 minutes of runtime
small	A quick diagnostics run that provides low fault detection
medium	A medium length diagnostics run that provides fault detection
large	A long length diagnostics run that provides high fault detection
complete	Runs all diagnostics, takes the longest time and provides the highest fault detection
servicecard	All service card-specific tests
nodeboard	All node board-specific tests
ioboard	All I/O board-specific tests
memory	Tests that target the memory subsystem on the computes
processor	Tests that target the compute processors
torus	Tests that target the torus network
power	Tests that target the power subsystem in node boards and I/O drawers
pcie	Tests that target the PCIe hardware in I/O drawers

Table 10-1 provides a list of the current test cases that are available, the average amount of run time in minutes, a brief description, and the buckets in which they run.

Table 10-1 Test cases

Name	Run time	Description	Bucket
initservicecard	1	Test performs a basic service card initialization.	servicecard
initnodeboard	1	Test performs a basic node board initialization	nodeboard
initioboard	1	Test performs a basic I/O board initialization	ioboard
envs	1	Test checks voltages, currents, temperatures, clocks and other environmental parameters on the node boards, I/O boards, and service cards.	small, medium, large, complete, servicecard, nodeboard, ioboard
bqlbist	2	Test runs Built-In Self-Test (BIST) on the BQL ASICS. Note: Do not run this test on link chips that are performing pass-through.	complete, nodeboard, ioboard

Name	Run time	Description	Bucket
bqlhssbist	2	Test runs HSS BIST on the BQL ASICs. Note: Do not run this test on link chips that are performing pass-through.	complete, nodeboard, ioboard
bqcbist	2	Test runs BIST on the BQC ASICs.	small, medium, large, complete, memory, processor, torus
bqchssbist	2	Test runs HSS BIST on the BQC ASICs.	small, medium, large, complete, processor, torus
powerclocks	1	Test performs a low-level, verbose POR sequence on the BQCs to ensure proper base function of the power, clocks and BQC.	
ddr_bitfail	6	Tests the A2 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, memory
ms_gen_short	5	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, memory
ms_gen_short_l2slice	5	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.	
ddr_error_stress	2	This test stresses the memory controller with different error conditions.	complete
coherency	2	Cache coherency test.	small, medium, large, complete, memory, processor
consistency	6	Cache consistency test.	small, medium, large, complete, memory, processor
qcd_mono	5	Tests the BQC QPX hardware on the compute nodes.	
dgemm	14	Tests the BPC ASIC, FPU, and memory subsystem on the compute nodes.	checkup, medium, large, complete, memory, processor
dgemm_pulse	14	Tests the BPC ASIC, FPU and the memory subsystem on the compute nodes using bursts of activity across the nodes exercising the power regulators.	checkup, medium, large, complete, nodeboard, ioboard, memory, power
trash	6	BQC random instruction stream consistency test.	large, complete, processor

Name	Run time	Description	Bucket
grub	6	BQC random instruction stream consistency test.	large, complete, processor
upc	3	Tests the BQC UPC hardware on the compute nodes.	large, complete, processor
msg_ran_diag1	5	Tests the BQC network hardware in loopback mode.	checkup, medium, large, complete, nodeboard, ioboard, torus
msg_diag_connectivity	5	Tests the BQC network hardware using torus functions.	checkup, medium, large, complete, nodeboard, ioboard, torus
qcd	10	Tests the BQC QPX hardware on the compute nodes.	large, complete, processor
boot	4	Basic boot block test.	complete, pcie
regulators	14	Stress test of the power regulators and their communications paths.	
regulators0	14	Stress test of the power regulators and their communications paths.	
regulators1	14	Stress test of the power regulators and their communications paths.	

10.3 Using the Navigator diagnostics interface

The Blue Gene Navigator administrative web application can be used to view the results of diagnostics runs, start a diagnostics run, and view the progress of current diagnostics runs. The Blue Gene Navigator is described in Chapter 2, “Navigator” on page 7.

10.4 Running diagnostics

There are two methods to initiate diagnostics. The most common method is from the graphical user interface, Blue Gene Navigator. Starting a diagnostics run from the Blue Gene Navigator is described in section 2.2.12, “Diagnostics” on page 21.

To run diagnostics, you must have the authorities listed in section 10.1.1, “Requirements” on page 120.

You can also run diagnostics from the command line. The `rundiags.py` command is located in the `/bgsys/drivers/ppcfloor/diags/bin` directory. It uses the following syntax:

```
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 midplanes, I/O drawers, or blocks on which to run the diagnostic test cases. The command uses the following switches:

- midplanes x,y,z...** List of midplanes on which to run diagnostics (for example, R00-M0, R00-M1, and R01-M0)
- io x,y,z...** List of I/O drawers on which to run diagnostics (for example, Q14-I5, R00-IC)
- blocks x,y,z...** List of blocks on which to run diagnostics

The list of midplanes, I/O drawers or blocks is a comma-separated list and must not contain any white spaces. You must specify the **--blocks** switch by itself. It is not compatible with the **--io** or **--midplanes** switches. You can use midplanes and I/O drawers in the same command, for example:

```
rundiags.py --midplanes midplane_list --io io_list [OPTIONS ... ]
```

Table 10-2 shows the options that you can use with the **rundiags.py** command.

Table 10-2 *rundiags.py* options

Option	Default	Description
--buckets	Complete	Runs the specified bucket (see 10.2, "Diagnostic test cases" on page 120)
--csport	32031	Control System port
--csuser	current user	Control System user
--dbname	bgdb0	Name of the database (see the bg.properties file)
--dbpassword	xxxxxx	Database password (see bg.properties file)
--dbschema	bgqsysdb	Name of the database schema (see bg.properties file)
--dbuser	bgqsysdb	Name or the database user (see bg.properties file)
--deleteblock	false	Diagnostics will delete the blocks when done
--disruptmasterclock	false	Allows the card that controls the master clock card to be re-initialized
--disruptiorack	false	Allows the I/O drawer that controls the I/O rack to be re-initialized
--floor	/bgsys/drivers/ppcfloor	Specifies an alternate floor directory
--forceblockfree	false	Forces a free of the diagnostics block prior to use
--help		Displays help text and exits
--insertras	false	Indicates whether diagnostics RAS should be inserted into the database
--killjobs	false	Kills any jobs or free any blocks that might prevent the run from proceeding
--mcserverport	1206	mcServer port
--midplanesperproc	2	The number of concurrent midplane runs allowed per service node processor
--midplanesperrow	16	The number of concurrent midplane runs allowed per row
--nodeleteblock	true	Diagnostics will not delete the block when done

Option	Default	Description
<code>--nogenblock</code>	false	Prevents diagnostics from creating blocks automatically
<code>--nokilljobs</code>	true	Does not kill any jobs or free any blocks that might prevent the run from proceeding
<code>--outputdir</code>	<code>/bgsys/logs</code>	Overrides the default output directory
<code>--properties</code>	<code>/bgsys/local/etc/bg.properties</code>	bg.properties file
<code>--savealloutput</code>	false	Saves all the output generated by diagnostics
<code>--sn</code>	localhost	Name of the service node
<code>--stoponerror</code>	false	Causes the run to abort when the first error is encountered
<code>--tests</code>	Run all tests	Allows you to provide a list of tests to be included in the run. Can be used in conjunction with <code>--buckets</code> to run additional test cases
<code>--testproperties</code>	test.properties	Diagnostics tests file
<code>--thresholds</code>	<code>/bgsys/drivers/ppcfloor/diags/etc/prod.properties</code>	prod.properties file
<code>--verbosity</code>	0	Verbosity level of output generated by diagnostics

10.5 Preventive maintenance

It is recommended that you run the diagnostics checkup bucket on each midplane at least once per month. This preventive maintenance provides coverage to determine any hardware problems while maintaining a minimum diagnostics runtime.

The checkup bucket might fail for a particular hardware problem, but you can obtain more in-depth analysis by running a more complete diagnostics bucket. Run the checkup bucket to find any failing hardware, and if you need further diagnosis, run other diagnostics. The diagnostics that you run depends highly on the specific failure.

10.6 Running diagnostics through a scheduler

To run diagnostics with a scheduler, you must create the diagnostics blocks ahead of time. Create the diagnostics blocks on a per-midplane basis, and name them with the diagnostics naming convention `_DIAGS_Rxx-Mx`. Using this naming convention, an example diagnostics command line is as follows:

```
rundiags.py --midplanes R00-M0 --nogenblock --bucket checkup
```

You must run the command from `/bgsys/drivers/ppcfloor/diags/bin` by user ID `bgqadmin` on the service node.

10.6.1 Running diagnostics from a LoadLeveler job

Example 10-1 is a sample LoadLeveler job command file for running diagnostics. The job class `diags` is defined only on the service node to guarantee that the job can be started only on the service node as required. The `--forceblockfree` option of the `rundiags.py` command must be specified because LoadLeveler will always boot the block.

Example 10-1 Job command file

```
bgqadmin@bgqsn2:> cat rundiag_R01-M1.cmd
# @ job_name = rundiags
# @ error = $(job_name).$(jobid).out
# @ output = $(job_name).$(jobid).out
# @ environment = COPY_ALL;
# @ wall_clock_limit = 00:30:00,00:25:00
# @ notification = error
# @ notify_user = bgqadmin
# @ job_type = bluegene
# @ bg_block = _DIAGS_R01-M1
# @ class = diags
# @ queue
/bgsys/drivers/ppcfloor/diags/bin/rundiags.py --forceblockfree --midplanes R01-M1 --test
envs --nogenblock
```

After submitting the job command file (in Example 10-1) to LoadLeveler and after the job runs, the output shown in Example 10-2 is generated.

Example 10-2 Output file

```
Blue Gene Diagnostics version 1.8.1 running on Linux 2.6.32-220.el6.ppc64 ppc64,
bgqsn2.rchland.ibm.com:9.5.45.102, built on 02-06-2012 01:21:44 by bgbuild, compile 0.
Blue Gene Diagnostics initializing...
Blue Gene Diagnostics starting...
Diagnostics run parameters:
    Run ID:                1202061701237140
    Command line arguments: --forceblockfree --midplanes R01-M1 --test envs
--nogenblock
    Host name:             bgqsn2.rchland.ibm.com
    Environment:          NORMAL
    Available processors:  16
    Maximum available memory: 3.906 GiB
    Process ID:           27140
    SN address:           127.0.0.1
    SN Control System port: 32031
    mcServer port:       1206
    Control System user:  bgqadmin
    Database name:        BGDBO
    Database schema:     bgqsysdb
    Database user:       bgqsysdb
    Test properties file: /bgsys/drivers/ppcfloor/diags/etc/tests.properties
    BG properties file:  /bgsys/local/etc/bg.properties
    HUP override enabled: true
    Output directory:    /bgsys/logs/BGQ/diags/120206_170123_27140
    User specified output dir: false
    Verbosity:           0
    Stop on first error: false
    Save all output:     false
    Generate block:      false
    Force block free:    true
    Floor directory:     /bgsys/drivers/ppcfloor
```



```

Poll hardware:           true
Hardware polling interval: 10
Hardware cooldown time: 60
Perform CE thresholding: true
Locations:               R01-M1
Tests to run:           envs

```

Running envs, iteration 1, on block _DIAGS_R01-M1. Results stored in
 /bgsys/logs/BGQ/diags/120206_170123_27140/envs__DIAGS_R01-M1_170139369/
 envs summary:

```

=====
Run ID:                  1202061701237140
Block ID:                _DIAGS_R01-M1
Start time:              Feb 06 17:01:39
End time:                Feb 06 17:02:03
Testcase:                envs
Iteration:               1
Passed:                  19
Marginal:                0
Failed:                  16
  R01-M1-N00 (SN 74Y7331YL30K1244032): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
  R01-M1-N01 (SN 74Y7331YL30K1231001): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
  R01-M1-N02 (SN 74Y7331YL30K1244002): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
  R01-M1-N03 (SN 74Y7331YL30K1244014): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
  R01-M1-N04 (SN 74Y7331YL30K124400D): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
  R01-M1-N05 (SN 74Y7331YL30K1216004): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
  R01-M1-N06 (SN 74Y7331YL30K124400A): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
  R01-M1-N07 (SN 74Y7331YL30K1216007): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
  R01-M1-N08 (SN 74Y7331YL30K124400C): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
  R01-M1-N09 (SN 74Y7331YL30K124401C): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
  R01-M1-N10 (SN 74Y7331YL30K1214001): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
  R01-M1-N11 (SN 74Y7331YL30K1244016): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
  R01-M1-N12 (SN 74Y7331YL30K1244029): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
  R01-M1-N13 (SN 74Y7331YL30K1244031): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
  R01-M1-N14 (SN 74Y7331YL30K1244009): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
  R01-M1-N15 (SN 74Y7331YL30K124401A): Node board link group 0 has an asserted alert signal.
    1 x Node board link group 0 has an asserted alert signal.
Unknown:                 0
Hardware status:         failed
Internal test failure:   false

```

Output directory:

/bgsys/logs/BGQ/diags/120206_170123_27140/envs__DIAGS_R01-M1_170139369/
 =====

Diagnostics log directory: /bgsys/logs/BGQ/diags/120206_170123_27140/
 Blue Gene diagnostics finished.

10.7 Diagnostics performance considerations

The performance of the service node and the service network can directly affect the diagnostics results. Running diagnostics on too many midplanes or performing too much work on the service node during diagnostics runs can cause diagnostics time outs, usually resulting in an UNKNOWN hardware status.

To cope with various hardware configurations and system usage scenarios, the diagnostics harness includes a small scheduler to pace the number of midplanes that are running at any given time. By default, the diagnostics harness only allows two midplanes per service node processor and 16 midplanes per row to run concurrently. Using large service nodes or reducing the amount of service node traffic during diagnostics from other midplanes (usually associated with booting other midplanes) can allow you to test more midplanes concurrently.

The Navigator web page for starting a new diagnostics run includes two options that you can use to customize the number of concurrent midplane runs:

- ▶ `--midplanesperproc`
- ▶ `--midplanesperrow`

These options are also the command-line arguments that you use to provide the same function.

The `--midplanesperproc` value describes the maximum number of midplane runs that are allowed to run in parallel per processor that is installed in the service node. This option helps prevent the service node from being overtaxed by processing that is required for diagnostics, which can cause time outs in the tests. Thus, if you specify `--midplanesperproc 2` and the service node has six processors installed, 12 midplane runs are allowed to run at a time if the `--midplanesperproc` value is the limiting factor.

The `--midplanesperrow` value can also limit the number of midplanes that are allowed to run at one time. This option describes the number of midplanes that are allowed to run within the same row. The row can also be a limiting factor for diagnostics in that the service network bandwidth is shared on a row-by-row basis. Spreading the diagnostics runs across the rows helps to prevent service network bottlenecks.



BGmaster

The Blue Gene/Q distributed process manager, known as BGmaster, consists of multiple components: `bgmaster_server`, `bgagentd`, and several client commands for interacting with `bgmaster_server`. BGmaster is responsible for managing the distributed `mc_server` components and all Control System components.

11.1 BGmaster

BGmaster is the process manager on Blue Gene/Q for the Control System servers. It acts as a centralized failure and recovery process. It can control the starting and stopping of various components including:

- ▶ Machine controller (MC) server
- ▶ SubnetMc
- ▶ Midplane Management Control System (MMCS) server
- ▶ Blue Gene Web Services (BGWS) server
- ▶ runjob server
- ▶ runjob mux
- ▶ Real-time server
- ▶ Toolkit for Event Analysis and Logging (TEAL)

There are multiple components that make up BGmaster: `bgmaster_server`, `bgagentd` and a set of client commands.

The `bgmaster_server` does not run binaries directly, leaving that task to the `bgagentd` processes, which makes it possible for managed binaries to continue running even if the `bgmaster_server` process stops. This lends robustness to the system. When `bgmaster_server` is restarted, the `bgagentd` processes register themselves again and report the binaries that are running to `bgmaster_server`. Because the intelligence in the system is in `bgmaster_server`, failover and restart policies do not run and client programs cannot do anything until it starts back up. The BGmaster client commands serve as both a user interface and as a scriptable interface to `bgmaster_server`.

Binaries managed by `bgmaster_server` are assigned a unique identifier consisting of the IP address of the agent running them followed by a colon and the process identifier assigned to the binary by the operating system (see Example 11-1 on page 132). Agents have similar unique identifiers consisting of the IP address and port that they are known by to `bgmaster_server`. These identifiers are used by several of the client commands to specify agents and managed binaries.

11.2 Running `bgmaster_server`

The process control daemon for BGmaster is `bgmaster_server`. It manages failover and restart policies and instructs the `bgagentd` processes to run managed servers under those policies. This section contains a basic guide and describes the commands used to start, manage, and stop the components of BGmaster and the binaries it manages.

Issuing the command `master_start bgmaster` starts `bgmaster_server` and all configured binaries. When `bgmaster_server` starts, it waits for agents to connect and register themselves. After it is registered, `bgmaster_server` instructs the agents to start running any binaries they are assigned.

To ensure that only one instance of `bgmaster_server` is running on a service node at a time, when `bgmaster_server` is started it writes a lock file in `/tmp` called `bgmaster_server-lock`. If a lock file already exists when `bgmaster_server` starts, it aborts immediately. The only contents of the file is the process identifier of `bgmaster_server`. When `bgmaster_server` exits, it removes the lock file. If `bgmaster_server` is abnormally aborted and cannot clean up the lock file, it must be manually removed before another `bgmaster_server` can be started.

Configuration for `bgmaster_server` comes from the `bg.properties` file. Client commands also take a `--properties` argument. The properties file passed to client commands is used only to provide parameters to the client command and is not forwarded to `bgmaster_server`. Changing the `bgmaster_server` properties file is accomplished with the `bgmaster_server_refresh_config` command.

Note that `bgmaster_server` can run under any user that has write access to the log directories (see Chapter 20, “Logging” on page 219) and read access to the administrative private key (see section 18.2.4, “Certificate file permissions” on page 208).

The following sections describe the commands available to manage and monitor BGmaster.

11.2.1 The `master_start` command

BGmaster, the binaries for any configured alias, or all configured binaries are started using the `master_start` command, which is located in the `/bgsys/drivers/ppcfloor/hlcs/sbin` directory.

The `master_start` command uses the following syntax:

```
master_start [alias | “bgmaster” | “binaries”] [OPTIONS ... ]
```

The command accepts the following arguments:

alias	A server alias, as defined in the <code>bg.properties</code> file. To use the alias argument, the <code>bgmaster_server</code> must be running, and a <code>bgagentd</code> must be running on the host associated with the alias.
bgmaster	Start the <code>bgmaster_server</code> process.
binaries	Start the managed binaries specified in the default <code>bg.properties</code> file or the location specified by <code>BG_PROPERTIES_FILE</code> environment variable if defined. The <code>bgmaster_server</code> and the associated <code>bgagentd</code> processes must be running to accept binaries as an argument.
--host host:pair	TCP/IP host and port pair to use to connect to <code>bgmaster_server</code> . This overrides the host option in the <code>[master.client]</code> section of the <code>bg.properties</code> configuration file.
--properties	This option provides the path to an alternate Blue Gene configuration file. The default file, <code>/bgsys/local/etc/bg.properties</code> or the location specified by <code>BG_PROPERTIES_FILE</code> environment variable if defined is used if the option is not specified.
--help	Prints help text.

11.2.2 The `master_status` command

To determine the current status of `bgmaster_server`, the `master_status` command is used. The `master_status` command returns the process identifier of `bgmaster_server`, a list of aliases not currently running, a list of connected agents, and the binaries they are running.

The `master_status` command is located in the `/bgsys/drivers/ppcfloor/hlcs/bin` directory.

Sample output from the command is shown in Example 11-1 on page 132.

The `master_status` command uses the following syntax:

```
master_status [OPTIONS ... ]
```


The command accepts the following arguments:

- | | |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| binary id | The binary identifier is a combination of the IP address and process identifier (PID), separated by a colon, of the system the binary is running on. |
| --host host:pair | TCP/IP host and port pair to use to connect to bgmaster_server. This overrides the host option in the [master.client] section of the bg.properties configuration file. |
| --properties | This option provides the path to an alternate Blue Gene configuration file. The default file, /bgsys/local/etc/bg.properties or the location specified by BG_PROPERTIES_FILE environment variable if defined is used if the option is not specified. |
| --help | Prints help text. |

Example 11-2 shows the status of the binary with the identifier of 172.20.2.38:27024.

Example 11-2 binary_status command output

```
$ ./binary_status 172.20.2.38:27024
172.20.2.38:27024|runjob_mux_fen8|RUNNING|2011-Oct-17 07:54:54
```

11.2.4 The bgmaster_server_refresh_config command

The **bgmaster_server_refresh_config** command updates bgmaster_server's configuration with a new or changed properties file. If no properties file is given, it uses the default. Updates to BGmaster policies are additive only. Existing policy parameters cannot be changed, deleted, or replaced, but new ones can be added.

The **bgmaster_server_refresh_config** command is located in the /bgsys/drivers/ppcfloor/hlcs/sbin directory.

The **bgmaster_server_refresh_config** command uses the following syntax:

```
bgmaster_server_refresh_config [filename] [OPTIONS ... ]
```

The command accepts the following arguments:

- | | |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| filename | This option provides the path to an alternate Blue Gene configuration file. The default file, /bgsys/local/etc/bg.properties or the location specified by BG_PROPERTIES_FILE environment variable if defined is used if the option is not specified. |
| --host host:pair | TCP/IP host and port pair to use to connect to bgmaster_server. This overrides the host option in the [master.client] section of the bg.properties configuration file. |
| --help | Prints help text. |

11.2.5 The binary_wait command

The **binary_wait** command will wait and not return until the specified binary fails. One use of the **binary_wait** command is to script a “watchdog” that can, for example, email a system administrator if a problematic server goes down.

The **binary_wait** command is located in the /bgsys/drivers/ppcfloor/hlcs/bin directory.

The syntax of the command is:

```
binary_wait [binary id] [OPTIONS ... ]
```

The command accepts the following arguments:

binary id	The binary identifier is a combination of the IP address and process identifier (PID), separated by a colon, of the system the binary is running on.
--host host:pair	TCP/IP host and port pair to use to connect to bgmaster_server. This overrides the host option in the [master.client] section of the bg.properties configuration file.
--properties	This option provides the path to an alternate Blue Gene configuration file. The default file, /bgsys/local/etc/bg.properties or the location specified by BG_PROPERTIES_FILE environment variable if defined is used if the option is not specified.
--help	Prints help text.

11.2.6 The fail_over command

The **fail_over** command forces a specific binary identifier to be stopped and restarted on another node. This behavior is controlled by the configured policy.

The **fail_over** command is located in the /bgsys/drivers/ppcfloor/hlcs/sbin directory.

The syntax of the command is:

```
fail_over [binary id] [OPTIONS ...]
```

The command accepts the following arguments:

binary id	The binary identifier is a combination of the IP address and process identifier (PID), separated by a colon, of the system the binary is running on.
--trigger (b/k/[a])	Trigger (binary, killed, agent) that will be used to find the associated policy for failover.
--host host:pair	TCP/IP host and port pair to use to connect to bgmaster_server. This overrides the host option in the [master.client] section of the bg.properties configuration file.
--properties	This option provides the path to an alternate Blue Gene configuration file. The default file, /bgsys/local/etc/bg.properties or the location specified by BG_PROPERTIES_FILE environment variable if defined is used if the option is not specified.
--help	Prints help text.

11.2.7 The alias_wait command

The **alias_wait** command signals BGmaster to wait for a binary that is associated with the specified alias to become active. This is a blocking wait with an optional timeout.

The **alias_wait** command is located in the /bgsys/drivers/ppcfloor/hlcs/bin directory.

The syntax of the command is:

alias_wait [**alias**] [**OPTIONS ...**]

The command accepts the following arguments:

alias Specifies the binary alias to wait on.

--timeout Specifies the number of seconds to wait for the specified binary to become active.

--host host:pair TCP/IP host and port pair to connect to bgmaster_server. This overrides the host option in the [master.client] section of the bg.properties configuration file.

--properties This option provides the path to an alternate Blue Gene configuration file. The default file, /bgsys/local/etc/bg.properties or the location specified by BG_PROPERTIES_FILE environment variable if defined is used if the option is not specified.

--help Prints help text.

11.2.8 The list_agents command

The **list_agents** command returns a list of identifiers for active agents and their associated running binaries. The return format is agentid::binaryid. An agent identifier is made up of the IP address from which the agent is running, a colon, and a port number that bgmaster_server uses to communicate with it. A binary identifier is made up of the IP address on which the binary is running, a colon, and its process identifier (PID).

The **list_agents** command is located in the /bgsys/drivers/ppcfloor/hlcs/bin directory.

The syntax of the command is:

list_agents [**OPTIONS ...**]

The command accepts the following arguments:

--fancy The output from the **list_agents** command will be labeled and formatted as shown in Example 11-3. This option is used by default.

--host host:pair TCP/IP host and port pair to use to connect to bgmaster_server. This overrides the host option in the [master.client] section of the bg.properties configuration file.

--properties This option provides the path to an alternate Blue Gene configuration file. The default file, /bgsys/local/etc/bg.properties or the location specified by BG_PROPERTIES_FILE environment variable if defined is used if the option is not specified.

--help Prints help text.

Example 11-3 Output from list_agents command

```
./list_agents
```

Agent host	Agent id (IP:Port)
-----	-----
bgqfen6-ib.rchland.ibm.com	172.20.2.36:50812
bgqfen7-ib.rchland.ibm.com	172.20.2.37:48920
bgqssn4-svc.rchland.ibm.com	192.169.3.14:52474
bgqssn5-svc.rchland.ibm.com	192.169.3.15:36874
bgqsn2-svc	192.169.3.2:49243

11.2.9 The `list_clients` command

The `list_clients` command returns a list of client identifiers for active clients. A client identifier consists of the IP address on which the client is running, a colon, and the port that `bgmaster_server` uses to communicate with it.

The `list_clients` command is located in the `/bgsys/drivers/ppcfloor/hlcs/bin` directory.

The syntax of the command is:

```
list_clients [OPTIONS ... ]
```

The command accepts the following arguments:

- host host:pair** TCP/IP host and port pair to connect to `bgmaster_server`, which overrides the host option in the `[master.client]` section of the `bg.properties` configuration file.
- properties** This option provides the path to an alternate Blue Gene configuration file. The default file, `/bgsys/local/etc/bg.properties` or the location specified by `BG_PROPERTIES_FILE` environment variable if defined is used if the option is not specified.
- help** Prints help text.

11.2.10 The `master_stop` command

The `master_stop` command can stop any or all controlled processes, any or all `bgagentd` processes and `bgmaster_server`. By default, it stops `bgmaster_server` and all managed binaries. Processes are allowed an orderly completion with a timeout before they are forced to end. The `bgmaster_only` option ends the `bgmaster_server` but none of the agents or managed binaries. By default, the initial signal is `SIGTERM`.

When `master_stop` is run in the form `master_stop bgmaster`, it stops `bgmaster_server` after telling it to end all managed binaries. The `bgagentd` processes will continue running. When `bgmaster_server` is started again, the `bgagentd` processes reconnect, and `bgmaster_server` instructs them to start the managed binaries. If you want to stop `bgmaster_server` only and not the managed binaries, use the `bgmaster_only` option of `master_stop`. When `bgmaster_server` is started again, the `bgagentd` processes will contact `bgmaster_server` and update it with the status of all of their managed binaries.

Stopping binaries under control of `bgmaster_server` can be accomplished by specifying them by binary identifier or by alias name. Specifying them by alias name is the simplest method and requires running the `master_stop <alias>` command. To use the binary identifier, use the `master_stop --binary <identifier>` command.

The `master_stop` command is located in the `/bgsys/drivers/ppcfloor/hlcs/sbin` directory.

The syntax of the command is:

```
master_stop [alias | "bgmaster" | "binaries" | "bgmaster_only"] [OPTIONS ... ]
```

The command accepts the following arguments:

- alias** Stops the binaries associated in the properties file with the specified alias.
- bgmaster** Stops the `bgmaster_server` process and all managed binaries.
- binaries** Stops all binaries managed by `bgmaster`.

bgmaster_only	Stops the bgmaster_server process but leaves the agents and managed binaries running. The bgmaster_server can then be restarted with master_start , and all agents report their status and the status of their managed binaries.
--binary	Stops a specific binary identifier. An alias can, according to policy, be associated with more than one instance of a binary, allowing the user to specify a particular instance by binary identifier.
--agent	Specifies a bgagentd process to stop by agent identifier. Using <i>All</i> instead of an identifier stops all bgagentd processes.
--signal	Sends the specified process this signal number and waits before forcing it to end. Note that bgmaster_server interprets the user specifying the --signal option as an intentional action. Therefore, no failover or restart policy is triggered. The expectation is that the user sent the signal because they wanted the process ended.
--host host:pair	TCP/IP host and port pair to connect to bgmaster_server to override the host option in the [master.client] section of the bg.properties configuration file.
--properties	Provides the path to an alternate Blue Gene configuration file. The default file, /bgsys/local/etc/bg.properties or the location specified by the BG_PROPERTIES_FILE environment variable, if defined, is used if the option is not specified.
--help	Prints help text.

11.2.11 The get_errors command

The **get_errors** command returns the contents of bgmaster_server's log ring buffer. The log ring buffer contains up to the last fifty logged error messages. These errors also generate RAS messages in the Blue Gene/Q database.

The **get_errors** command is located in the /bgsys/drivers/ppcfloor/hlcs/bin directory.

The syntax of the command is:

```
get_errors [OPTIONS ... ]
```

The command accepts the following arguments:

--host host:pair	TCP/IP host and port pair to connect to bgmaster_server to override the host option in the [master.client] section of the bg.properties configuration file.
--properties	Provides the path to an alternate Blue Gene configuration file. The default file, /bgsys/local/etc/bg.properties or the location specified by BG_PROPERTIES_FILE environment variable, if defined, is used if the option is not specified.
--help	Prints help text.

11.2.12 The get_history command

The **get_history** command returns the contents of bgmaster_server's history log ring buffer. The history log contains up to the last fifty instances of process starts and stops since bgmaster_server was started.

The `get_history` command is located in the `/bgsys/drivers/ppcfloor/hlcs/bin` directory.

The syntax of the command is:

```
get_history [OPTIONS ... ]
```

The command accepts the following arguments:

- host host:pair** TCP/IP host and port pair to connect to `bgmaster_server`. This overrides the host option in the `[master.client]` section of the `bg.properties` configuration file.
- properties** Provides the path to an alternate Blue Gene configuration file. The default file, `/bgsys/local/etc/bg.properties` or the location specified by `BG_PROPERTIES_FILE` environment variable, if defined, is used if the option is not specified.
- help** Prints help text.

11.2.13 The `monitor_master` command

The `monitor_master` command provides a rolling update of events and errors from `bgmaster_server`. When `monitor_master` makes an initial connection to `bgmaster_server`, it gathers the contents of the history and error buffers and prints them out. The `monitor_master` command maintains the connection, and `bgmaster_server` sends each new event and error back to `monitor_master`, which displays them on standard out. The command can be stopped from the command line by the Ctrl-C keyboard sequence.

The `monitor_master` command is located in the `/bgsys/drivers/ppcfloor/hlcs/bin` directory.

The syntax of the command is:

```
monitor_master [OPTIONS ... ]
```

The command accepts the following arguments:

- host host:pair** TCP/IP host and port pair to connect to `bgmaster_server`. This overrides the host option in the `[master.client]` section of the `bg.properties` configuration file.
- properties** This option provides the path to an alternate Blue Gene configuration file. The default file, `/bgsys/local/etc/bg.properties` or the location specified by `BG_PROPERTIES_FILE` environment variable, if defined, is used if the option is not specified.
- help** Prints help text.

11.3 The `bgagent` daemon system process watcher

The `bgagent` daemon (`bgagentd`) is the BGmaster system's process watcher. The `bgagent` daemon operates solely on instructions from `bgmaster_server`. When `bgagentd` is started, it periodically attempts to connect to `bgmaster_server`. In a standard configuration, the `bgagentd` daemon running on the service node connects to `bgmaster_server` on the host and port (typically `127.0.0.1:32041`) specified on the host parameter of the `[master.agent]` section of the `bg.properties` file. Agents running outside the service node are configured to connect to `bgmaster_server` using the IP address of the service node. To configure the IP address, use the `--host` option in the `bgagent` init script as described in 11.3.1, "Starting `bgagentd`".

11.3.1 Starting bgagentd

When the Blue Gene/Q software is installed, there is an init script named *bgagent* placed in */etc/init.d*. This init script runs at boot time to automatically start *bgagentd*. After it is started *bgagentd* runs continuously in the background. If you must stop, start, or restart *bgagentd*, use the *bgagent* script like any other. The script accepts the arguments: start, stop, status, restart, try-restart, and force-reload.

Additionally the *bgagent* init script can pass options to the **bgagentd** command, for example, **bgagentd** is passed the option `--users`, shown below, to inform the program which users it can run as.

```
OPTIONS="--users bgqadmin,bgws,bgqsysdb --workingdir $workingdir --host
172.20.3.1:32041"
```

The accepted arguments for the **bgagentd** command are:

--properties	Provides the path to an alternate Blue Gene configuration file. The default file, <i>/bgsys/local/etc/bg.properties</i> or the location specified by <i>BG_PROPERTIES_FILE</i> environment variable, if defined, is used if the option is not specified.
--logdir	Specifies a directory, other than the default <i>/bgsys/logs/BGQ</i> , to write the <i>bgagentd</i> logs to.
--debug true false	Debug mode sends the log output to the console and runs <i>bgagentd</i> as a foreground process.
--workingdir	Provides the current working directory for <i>bgagentd</i> . This parameter is usually set in the <i>bgagent</i> init script.
--host host:pair	TCP/IP host and port pair to connect to <i>bgmaster_server</i> . This overrides the host option in the <i>[master.agent]</i> section of the <i>bg.properties</i> configuration file. For agents that do not run on the service node this should be specified in the <i>bgagent</i> init script and match the service node IP address that <i>bgmaster_server</i> is listening on. For example, if <i>bg.properties</i> is configured where <i>[master.server]</i> has <i>agent_listen_ports = 127.0.0.1:32041,172.20.3.1:32041</i> then the <i>bgagent</i> init script <i>OPTIONS</i> string should include <code>--host 172.20.3.1:32041</code> .
--users	Provides a list of user identifiers to which <i>bgagentd</i> is permitted to switch. Specify "root" to disallow user identifier switching. This parameter is usually set in the <i>bgagent</i> init script. The requested user identifier is set in <i>bg.properties</i> under the <i>[master.agent]</i> section as <i>agentuid=</i> .
--help	Prints help text.

11.3.2 Stopping bgagentd

The *bgagentd* processes are intended to be started at system boot and not stopped until the system is shut down. There is, however, the capability to stop agents with the **master_stop** command (see 11.2.10, "The *master_stop* command" on page 136) to make it easier than logging in to each individual system and running the init script's stop routine. When running **master_stop --agent <agent identifier>**, *bgmaster_server* sends a request to the specified *bgagentd* asking it to end itself. Be aware that **master_start** cannot then start the *bgagentd* process back up. After the *bgagentd* process is down, *bgmaster_server* has no other contact with that node.

When a bgagentd process ends, it attempts to end all of its managed binaries to make sure they do not continue unmanaged. To guarantee that child processes get killed when the agent gets killed, agentuid in the properties file must be set as root and bgagentd must be started as root. This limitation might not be acceptable in your environment. If so, it is important that you not send signals that cannot be handled (such as SIGKILL) to bgagentd. Ending bgagentd must be done using the bgagent init script installed on the system.

11.4 Configuration

Configuration of the servers that BGmaster manages is centered around server instances represented by configuration *aliases*. An alias is a short name that represents a specific executable. Each alias has an associated policy, its pairs of triggers and behaviors, the number of instances that it can run, the hosts on which it can run, and a list of the binary identifiers associated with it. Figure 11-1 is a graphical representation of an alias and its associated concepts, including the list of configured hosts to use, the list of actively running binaries, and the policy with its associated trigger/behavior pairs.

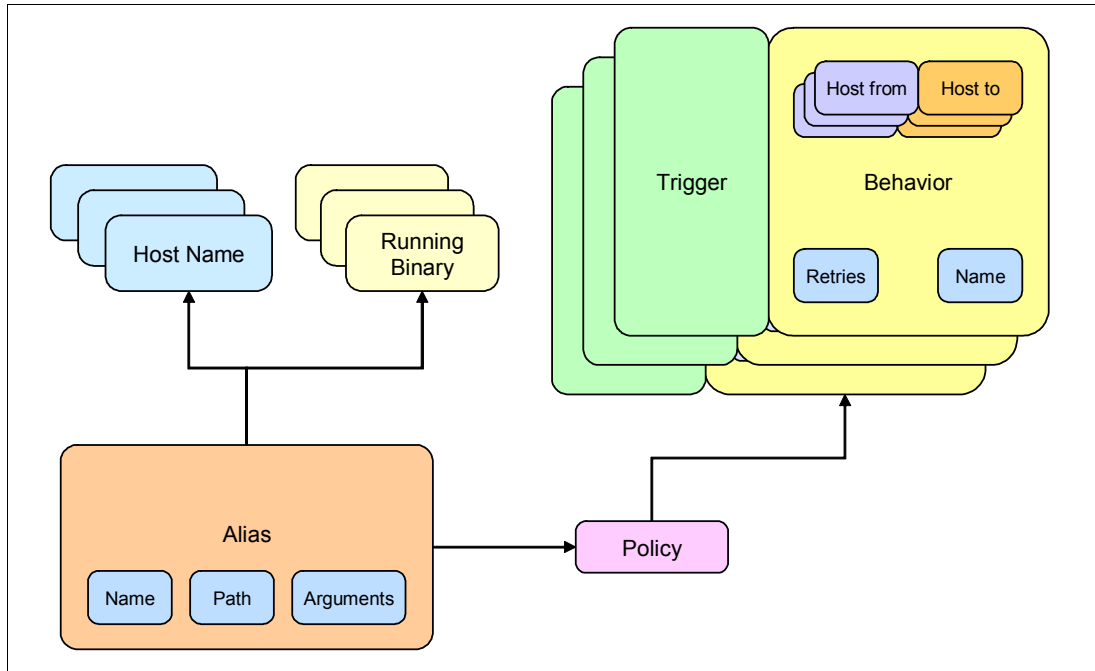


Figure 11-1 Alias concept

A *policy* is a set of triggers, their associated behaviors, and allowable number of instances. Policy configuration is flexible. Policies are stored in the bg.properties file. A trigger is a limited set of events that can cause a policy to enact a particular behavior. The list of acceptable triggers are *killed*, *binary* or *agent*:

- killed** Binary received a kill signal.
- binary** Binary abnormally ended.
- agent** The agent managing the binary failed.

Killed and binary are distinguished by the signal with which they fail. SIGTERM and SIGKILL, for example, are killed triggers. SIGSEGV and SIGABRT are considered binary triggers.

A *behavior* is an action, and instructions for that action to take when a trigger condition is detected. The components of a behavior are an action, a retry count, and an optional set of failover pairs. The supported actions are:

failover	Analyze the failover pairs and if the triggered binary is running on the first host in a pair, fail it over to the second.
restart	Attempts to restart the binary on the same host up to the number of times specified in the retry count parameter.
cleanup	Cleans up internal data structures and logs the error. (This is the default.)

11.4.1 Sample configuration

For this example, consider a Blue Gene/Q system with two front end nodes and two subnet service nodes. The subnet service nodes are assigned to IP addresses 172.16.1.135 and 172.16.1.136. The front end nodes are assigned to IP addresses 172.16.1.133 and 172.16.1.134. The front end nodes each need to run a runjob mux, and the subnet service nodes each need to run a SubnetMc. In this setup, each front end node and subnet service node must have an agent running that will connect to the `bgmaster_server` process running on a service node. Each agent will manage the starting and stopping of the binaries (runjob mux or SubnetMc) assigned to them through communications with the `bgmaster_server` process.

While multiple instances of a server binary (with the exact same configuration) can be run by setting the instance count and using a single alias, this is not typically done. The conventional method for running multiple instances of a server binary is to configure a unique alias name for each instance (for example, Subnet1, Subnet2, and so on).

When adding a front end node to the system, `bg.properties` sections like `[bgws]` and `[bg_console]` must be updated. Those `bg.properties` changes are outside the scope of this chapter. The following list outlines the main BGmaster sections of the `bg.properties` file that must be updated for this example system configuration:

- ▶ In the `[master.binmap]` section of the `bg.properties` file, define the aliases. The alias name for each SubnetMc process must match the *Name* keyword in the `[machinecontroller.subnet.X]` section. By default the first subnet name is "Subnet1", the second subnet name is "Subnet2", and so on. A unique alias name is used for each instance of a SubnetMc running on a subnet service node and each runjob mux running on a front end node. This alias is used because the aliases must be associated with different agents and this configuration determines the host system where the binary will run.

```
[master.binmap]
# Maps an alias to a binary path
...
bgmaster_server=/bgsys/drivers/ppcfloor/hlcs/sbin/bgmaster_server
Subnet1=/bgsys/drivers/ppcfloor/control/sbin/SubnetMc_64
Subnet2=/bgsys/drivers/ppcfloor/control/sbin/SubnetMc_64
runjob_mux1=/bgsys/drivers/ppcfloor/hlcs/sbin/runjob_mux
runjob_mux2=/bgsys/drivers/ppcfloor/hlcs/sbin/runjob_mux
...
```

If you have aliases (binaries) listed in the `[master.binmap]` section that you do not want started when the associated `bgagentd` starts up you either need to remove them or comment them out. If you comment out or delete an alias (binary) in the `[master.binmap]` section you will also need to comment out or delete the alias under the following `bg.properties` sections or `bgmaster_server` will fail with a configuration error when starting

up: [master.logdirs], [master.binargs], [master.user], [master.policy.host_list], [master.policy.instances] and [master.policy.map].

- In the [master.binargs] section of the bg.properties file, set up command-line arguments. In this case, different command-line arguments are needed for the two SubnetMc processes.

```
[master.binargs]
# List of arguments for each alias specified in [master.binmap] section
...
Subnet1=--ip 172.16.1.135,172.16.1.136 --inport 33456
Subnet2=--ip 172.16.1.136,172.16.1.135 --inport 33457
...
```

- Assign the aliases to hosts in the [master.policy.host_list] section. It is important to correctly match the alias with the host IP addresses where the binary can run. In this case, each SubnetMc has a preferred host where it can run and a backup (failover) host where it can run. Each runjob mux instance will be allowed to run on only one host and will have a restart policy configured but no failover policy.

```
[master.policy.host_list]
# Comma separated list of hosts on which the binary associated with the alias may start
....
Subnet1=172.16.1.135,172.16.1.136
Subnet2=172.16.1.136,172.16.1.135
runjob_mux1=172.16.1.133
runjob_mux2=172.16.1.134
...
```

- In the [master.policy.failure] section, define the policy. In this case, define a restart policy that is used by the runjob muxes and a failover policy for the SubnetMcs.

```
[master.policy.failure]
# Format: policy_name=trigger,[failover|restart|cleanup],<retries>,<failover_from:failover_to>|etc
...
server_restart=binary,restart,3
Subnet1_failover=agent,failover,3,172.16.1.135:172.16.1.136|172.16.1.136:172.16.1.135
Subnet2_failover=agent,failover,3,172.16.1.136:172.16.1.135|172.16.1.135:172.16.1.136
...
```

- Assign the policies to the alias in the [master.policy.map] section.

```
[master.policy.map]
# Map comma separated list of named policies to an alias
...
Subnet1=Subnet1_failover
Subnet2=Subnet2_failover
runjob_mux1=server_restart
runjob_mux2=server_restart
...
```

- When front end nodes or subnet service nodes are configured, bgmaster_server must be setup to listen for bgagent connections on the network interfaces that the agents use when connecting. This is configured by setting the *agent_listen_ports* option in the [master.server] section of the bg.properties file. The *agent_listen_ports* keyword contains a list of IP address:port pairs, separated by commas. Note that bgmaster_server will accept connections only on the IP addresses in the list.

To correctly configure the agent listen ports in bgmaster_server, you must first find the IP address associated with the network interface. The **ip addr show** command is used to find the IP address for the eth0 interface because, in this example case, the agents running on the front end nodes and subnet service nodes will connect to the service node over the 192.168 network:


```

bgqsn:~> ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 5c:f3:fc:05:46:20 brd ff:ff:ff:ff:ff:ff
    inet 192.168.3.103/16 brd 192.168.255.255 scope global eth0
    inet6 fe80::5ef3:fcff:fe05:4620/64 scope link
        valid_lft forever preferred_lft forever

```

Next, the *agent_listen_ports* keyword in the [master.server] section must be configured to listen to the IP address on that network. In this case, there is an agent running on the service node that connects over the IP address 127.0.0.1 (localhost) and also agents running on front end nodes and subnet service nodes that connect using the 192.168.3.103 IP address.

```

[master.server]
    # bgmaster_server configuration options
    ...
    agent_listen_ports = 127.0.0.1:32041,192.168.3.103:32041
    ...

```

11.4.2 BGmaster network configuration

BGmaster is highly configurable and consequently complex in some situations. It is important to consider the networking environment carefully and which processes are to be managed on which machines and how host names and IP addresses are to be apportioned. In a simple environment, a number of runjob mux processes are located off the primary service node. In more complex environments, several subnet service nodes and active backup service nodes are also running managed binaries.

It is important to understand how BGmaster components initiate their connections and identify themselves to understand how managed components get assigned to run in the proper place. The following list describes the sequence of making connections:

- ▶ When the bgagent daemon starts (see 11.3.1, “Starting bgagentd” on page 139 for information about startup) it repeatedly attempts to connect to the host and port specified by the *host* parameter in the [master.agent] section of the bg.properties file or the host and port indicated by the **bgagentd** `--host` argument. The bgagent daemon continues connection attempts until it successfully connects. Similarly, when the bgmaster_server starts it attempts to listen on all ports configured in the [master.server] section of bg.properties file as *agent_listen_ports* and *client_listen_ports*. As long as bgmaster_server can successfully bind and listen on at least one port of each type it will start.
- ▶ After bgagentd connects to bgmaster_server, bgagentd starts its own listener, and bgmaster_server connects back to bgagentd. These two connections are used by bgmaster_server and bgagentd for request/response protocols.
- ▶ The BGmaster client programs attempt to connect to bgmaster_server in much the same way as bgagentd. They use the *host* parameter in the [master.client] section of the bg.properties file to open an initial connection.
- ▶ Finally the managed binaries are assigned to hosts associated with their aliases in the [master.policy.host_list] section of the bg.properties file. Hosts are identified by one of their IP addresses or the host name associated with one of their IP addresses. Whether that host is available to start the binary depends upon whether a bgagentd has connected to bgmaster_server using that IP address.

11.5 Troubleshooting

The `bgmaster_server` maintains a ring buffer of start/stop events and errors. These ring buffers can be accessed with the `get_history` (see section 11.2.12, “The `get_history` command” on page 137) and `get_errors` (see section 11.2.11, “The `get_errors` command” on page 137) commands respectively. This data is also sent to the RAS database for permanent storage.

11.5.1 Configuration issues

The following items are some possible configuration problems that can lead to confusing or inconsistent behavior by `bgmaster_server` and `bgagent`:

- ▶ The `host` parameter in the `[master.agent]` section of the `bg.properties` file must point to at least one `agent_listen_port` in the `[master.server]` section.
- ▶ Because the agent is identified by the IP address it uses to connect to `bgmaster_server`, all binaries that it manages must be mapped to that IP address in the `[master.policy.host]` section of the `bg.properties` file.
- ▶ If a `bgagentd` can connect to `bgmaster_server` on multiple networks, it is identified by the first IP address that it can successfully use to connect. If the binaries that it needs to manage are only mapped to one of the IP addresses, and it is not the address that the agent uses to connect to `bgmaster_server`, those binaries cannot start.

Example 11-4 shows a `properties` file that can cause a subtle problem.

Example 11-4 bg.properties file options

```
[master.server]
  agent_listen_port=10.10.10.2:32041,192.168.1.2:32041
[master.agent]
  host=10.10.10.2:32041,192.168.1.2:32041
[master.policy.host]
  my_alias=10.10.10.2
```

In Example 11-4, if an agent connects to `bgmaster_server` on 10.10.10.2, `my_alias` starts successfully. If, however, the `bgagentd` connects on 192.168.1.2, `my_alias` does not start.

- ▶ If host names are used in `bg.properties` files, DNS and `/etc/hosts` files must map to the IP addresses that `bgagentd` processes to connect to `bgmaster_server`. Keep in mind that a single-service node might have a system-defined name and several other names mapped to multiple IP adapters using multiple IP addresses.
- ▶ Administrators should verify that the `host` setting in the `[master.agent]` section is configured correctly. The `host` option should be set such that the agent running on the service node only connects to `bgmaster_server` using the 127.0.0.1 interface. For agents running remotely, the init script needs to be configured to use the host where `bgmaster_server` is running. Modifying the agent init script is explained in section 11.3.1, “Starting `bgagentd`”. If these settings are misconfigured, under certain conditions, this can result in servers failing to start or failing to restart as expected.

Example 11-5 shows an incorrect setting for the `host` option.

Example 11-5 Incorrect setting for the host option in [master.agent] section

```
[master.agent]
  # bgagentd configuration options
  host=127.0.0.1:32041,192.168.3.103:32041
```

Example 11-6 shows the correct setting for the *host* option.

Example 11-6 Correct setting for the host option in [master.agent] section

```
[master.agent]
# bgagendtd configuration options
host=127.0.0.1:32041
```

- ▶ All servers which run on the service node must be configured to start on IP address 127.0.0.1 in the [master.policy.host_list] section. Under certain conditions, using *localhost* rather than 127.0.0.1 can result in servers failing to start or restart. All instances of *localhost* should be replaced with 127.0.0.1. Example 11-7 shows the correct setting for the Blue Gene Control System servers that run on the service node:

Example 11-7 Correct settings for the servers that run on the service node

```
[master.policy.host_list]
# Comma separated list of hosts which the binary associated with the alias may start
...
bgmaster_server = 127.0.0.1
bgws_server = 127.0.0.1
realtime_server = 127.0.0.1
runjob_server = 127.0.0.1
runjob_mux = 127.0.0.1
mc_server = 127.0.0.1
mmcs_server = 127.0.0.1
tea1_server = 127.0.0.1
...
```

11.5.2 RAS messages

BGmaster can generate many RAS messages. The complete list of BGmaster RAS messages can be viewed through Blue Gene Navigator by clicking the RAS book link under the Knowledge Center tab (see Chapter 2.2.15, “Knowledge Center” on page 27) and then clicking the BGMMASTER component link within the RAS book.

This information is also in the `/bgsys/drivers/ppcfloor/ras/etc/RasEventBook.htm` file.



Midplane Management Control System server

The Midplane Management Control System (MMCS) configures and allocates blocks of compute nodes and I/O nodes. As part of the *High-Level Control System (HLCS)*, MMCS runs on the service node and provides an interface to the hardware level system components. You can access the MMCS directly using the `bg_console` command or through the Scheduler APIs.

MMCS server is also referred to by its alternate name `mmcs_server` in this chapter.

In addition to the high-level control functions of the MMCS server, the following functions are incorporated into the server:

- ▶ A relay that passes Reliability, Availability, and Serviceability (RAS) messages from the MC Server to the database
- ▶ Environmental monitoring

12.1 MMCS components

This section describes the components of MMCS and how it serves as an interface between users and the Blue Gene/Q hardware. Users access MMCS primarily through a job scheduler, such as LoadLeveler. However, system administrators generally want to use the system's resources more directly through `bg_console` commands.

Figure 12-1 shows the Blue Gene/Q Control System architecture and the role of the MMCS server.

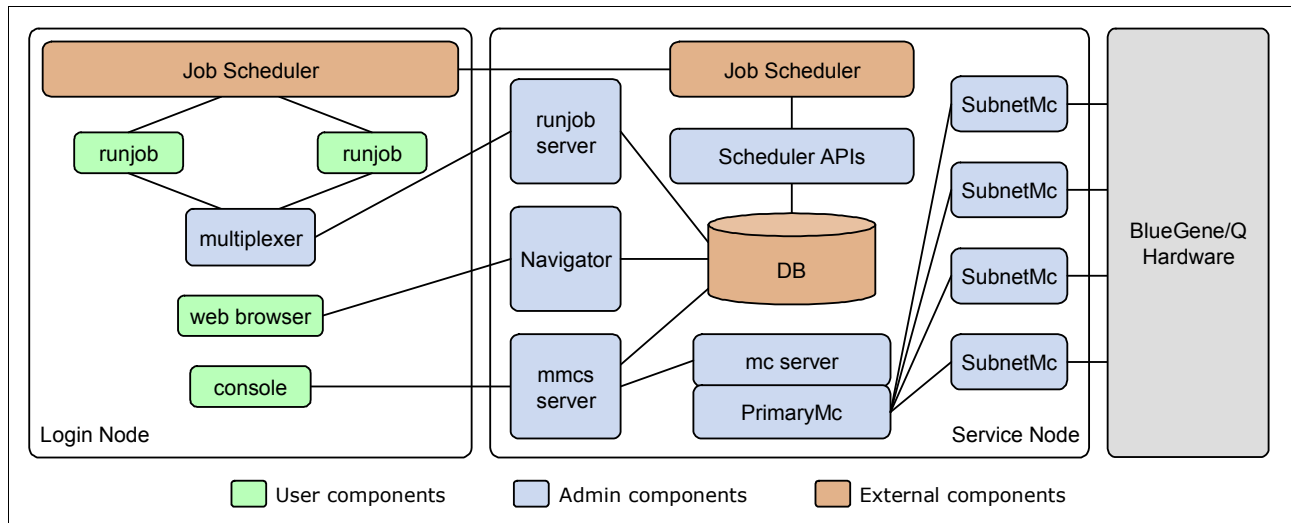


Figure 12-1 Control System architecture

12.1.1 MMCS server handles requests from clients

MMCS server, Figure 12-1, runs on the service node and listens for incoming connections from MMCS clients or script clients. MMCS commands are issued from `bg_console`. Scripting facilities are also provided through `bg_console`. MMCS server can handle multiple client connections. It performs other functions simultaneously, such as monitoring for RAS events and environmental monitoring.

12.1.2 Blocks are defined in the database

MMCS server allocates blocks configured in the central database. Blocks can be created by system administrators based on the needs of the users or dynamically by job schedulers, such as LoadLeveler. For more information, see Chapter 3, “Compute and I/O blocks” on page 29. MMCS server performs the actual block allocation, booting, link chip configuration, and deallocation based on requests from its clients. It keeps track of allocated blocks within the Blue Gene/Q system.

12.1.3 MC server communicates with the Blue Gene/Q hardware

MMCS server clients are not aware of the existence of MC server (mc server in Figure 12-1) in the Control System, but MC server plays a key role in the process. Through MC server, the MMCS server can monitor the availability of the hardware. MC server also controls access to the hardware to avoid conflicts. See section 13.1, “Machine controller server” on page 160 for more details.

12.2 Starting and stopping MMCS server

The MMCS server executable is `/bgsys/drivers/ppcfloor/hlcs/sbin/mmcs_server`. Starting and stopping MMCS server is controlled by BGmaster; however, it can be done interactively with the `master_stop` or `master_start` commands described in 11.2, “Running bgmaster_server” on page 130.

12.3 Configuration

MMCS server can be configured by setting options in the Blue Gene/Q configuration file or by setting command-line options when the server starts.

12.3.1 Configuration file options

The configuration options used by MMCS server are in the `[mmcs]` section and the `[mmcs.envs]` and `[mmcs.blockgovernor]` subsections of the Blue Gene/Q configuration file (`/bgsys/local/etc/bg.properties`). Configurable options for MMCS server include (default values are shown):

- ▶ `listen_ports = [::1]:32031,127.0.0.1:32031`
Defines the ports that MMCS server will listen for connections on. The format is a comma-separated list of host:port pairs. The host parameter is a TCP/IP address or host name. TCP/IP addresses containing colons must be enclosed in `[]`. Link local ipv6 addresses must include the interface name appended after a `%` character. The port parameter is a port number or service name. The host and port are optional. If the host is not specified MMCS server will listen on localhost. If the port is not specified the default port is used (32031).
- ▶ `shutdown_timeout = 90`
Number of seconds to wait for the I/O nodes to shut down.
- ▶ `wait_boot_free_time = 20`
Sets the maximum wait time, in minutes, for a boot to complete.
- ▶ `minimum_boot_wait = 360`
Minimum number of seconds to wait for a boot to complete before freeing the block.
- ▶ `reconnect_blocks = false`
Defines the default behavior when reconnecting to blocks booted in a previous instantiation of MMCS server.
- ▶ `log_dir = /bgsys/logs/BGQ`
Sets the default I/O log directory.
- ▶ `connection_pool_size = 10`
Defines the maximum number of connections held in the database connection pool.
When all the connections in the pool are used, it takes longer to connect and disconnect to the database. Proper tuning is important. This value should be set to the average number of concurrently booted I/O and compute blocks plus an additional ten connections to account for other `mmcs_server` database activity. For example, if on average one I/O block and four compute blocks are booted, set this value to 15.
- ▶ `cn_boot_slope = .017`

Linear boot timeout slope for compute blocks. Units for timeout slope are in seconds per node.

- ▶ `io_boot_slope = .017`

Linear boot timeout slope for I/O blocks. Units for timeout slope are in seconds per node.

The [mmcs.envs] section sets the environmental polling intervals. Specifying 0 turns off polling for that type of hardware. The values can be reset using the **refresh_config mmcs_server** command from `bg_console`. Using the **refresh_config** command allows you to change the values without stopping the `mmcs_server` process. The default values for the [mmcs.envs] section are shown in Example 12-1. More information about environmental polling can be found in section 12.5, “Environmental polling” on page 151.

Example 12-1 Default [mmcs.envs] values

```
sc_interval_seconds = 1800
io_interval_seconds = 300
nc_interval_seconds = 300
bulk_interval_seconds = 300
coolant_interval_seconds = 300
optical_interval_seconds = 3600
perfdata_interval_seconds = 300
```

The block governor manages the rate at which the MMCS server can poll the database. Configurable options for the [mmcs.blockgovernor] section include (defaults are shown):

- ▶ `max_concurrent = 0`
Number of block transactions that can be running at one time.
- ▶ `interval = 10`
Length of a measurement interval in seconds.
- ▶ `max_tran_rate = 5`
Number of block transactions that can be started in one interval.
- ▶ `repeat_interval = 15`
Minimum seconds between repeat of a transaction.

12.3.2 Command-line options

MMCS server (`mmcs_server` executable) accepts the following options on the command line:

- listen-ports <host:port,host:port>**
Listens on the comma-separated list of host:port pairs.
- properties**
Provides the path to an alternate Blue Gene/Q configuration file. The default file, `/bgsys/local/etc/bg.properties` or the location specified by `BG_PROPERTIES_FILE` environment variable if defined is used if the option is not specified.
- iolog**
Specifies the directory to log output from I/O nodes. The default directory is `/bgsys/logs/BGQ`.
- mcserverip**
Connects to `mc_server` on the specified TCP/IP address.
- mcserverport**
Connects to `mc_server` on the specified port.
- no-poll-db**
Does not poll the database for block actions.
Note: Do not use this option in a production environment because blocks can never boot if database polling is disabled.

--poll-db	Polls the database for block actions.
--no-poll-envs	Does not poll for environmental monitoring data.
--poll-envs	Polls for environmental monitoring data.
--no-reconnect	Do not reconnect blocks booted in a previous instantiation of mmcs_server.
--reconnect	Reconnect blocks booted in a previous instantiation of mmcs_server.
--shutdown-timeout	Sets the number of seconds to wait for I/O nodes to shut down.
--boot-timeout-and-free	Default number of minutes to wait for a boot to complete before freeing the block.
-h [--help]	Displays help text for the command.

Important: Keep in mind that MMCS server (mmcs_server) relies on MC server (mc_server) to maintain contact with the hardware. Therefore you must also start the mc_server manually if you are going to bypass BGMaster.

12.4 Mailbox output

Each node on an allocated block has a mailbox associated with it, and 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 MMCS server.

Mailbox output is primarily debug and command output from the I/O node kernel and the Compute Node Kernel. Application program output is handled separately.

MMCS server has various options for directing this mailbox output:

- ▶ Default
 - If you do nothing, the mailbox output is directed to the MMCS server log.
- ▶ Redirection
 - MMCS server has a **redirect** command that sends the mailbox output back to bg_console. The mailbox data is sent as it is received from the mailbox.

12.5 Environmental polling

MMCS server also performs environmental polling to monitor the status of the hardware. Monitoring on Blue Gene/Q 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 in debugging problems. By default, environmental data is collected at various intervals depending on the hardware. Current and historical data can be viewed in Blue Gene/Q Navigator, which is discussed in Chapter 2, “Navigator” on page 7. To assist in the efficient administration of the system, environmental polling also generates RAS events when abnormal readings are encountered.

12.5.1 How the environmental monitor works

Environmental polling starts when the MMCS server is started. The types of hardware monitored are service cards, I/O boards, and node boards. Performance data, optical data, bulk power data, Health Check data, and coolant monitoring data are also collected by MMCS server.

There are situations that might arise when you want to disable the monitoring functions temporarily. As a general rule, before changing or disabling the monitor, check with IBM Support.

12.5.2 Polling intervals

The current polling intervals are in the [mmcs.envs] section of the `/bgsys/local/etc/bg.properties` file. They are:

Service cards	<code>sc_interval_seconds</code>
I/O boards	<code>io_interval_seconds</code>
Node boards	<code>nc_interval_seconds</code>
Bulk Power Modules	<code>bulk_interval_seconds</code>
Coolant monitors	<code>coolant_interval_seconds</code>
Optical modules	<code>optical_interval_seconds</code>
Performance data	<code>perfdata_interval_seconds</code>

The following list provides the default polling interval and the range of supported values that can be used:

- ▶ Service cards: Defaults to 30 minute interval. The interval can be specified using the property `sc_interval_seconds` in the `bg.properties` file. The interval can range from 60 to 3600 seconds (1 to 60 minutes) or set to 0 to turn off polling.
- ▶ I/O boards: Defaults to five minute interval. The interval can be specified using the property `io_interval_seconds` in the `bg.properties` file. The interval can range from 60 to 3600 seconds (1 to 60 minutes) or set to 0 to turn off polling.
- ▶ Node boards: Defaults to five minute interval. The interval can be specified using the property `nc_interval_seconds` in the `bg.properties` file. The interval can range from 60 to 3600 seconds (1 to 60 minutes) or set to 0 to turn off polling.
- ▶ Bulk Power Modules: Defaults to five minute interval. The interval can be specified using the property `bulk_interval_seconds` in the `bg.properties` file. The interval can range from 60 to 3600 seconds (1 to 60 minutes) or set to 0 to turn off polling.
- ▶ Coolant monitors: Defaults to five minute interval. The interval can be specified using the property `coolant_interval_seconds` in the `bg.properties` file. The interval can range from 60 to 3600 seconds (1 to 60 minutes) or set to 0 to turn off polling.
- ▶ Optical modules: Defaults to 60 minute interval. The interval can be specified using the property `optical_interval_seconds` in the `bg.properties` file. The interval can range from 60 to 3600 seconds (1 to 60 minutes) or set to 0 to turn off polling.
- ▶ Performance data: Defaults to five minute interval. The interval can be specified using the property `perfdata_interval_seconds` in the `bg.properties` file. The interval can range from 60 to 1800 seconds (1 to 30 minutes) or set to 0 to turn off polling. The data being collected by MMCS server is Control System performance data.
- ▶ Health Check: Always runs on a five minute interval and cannot be changed or turned off. MMCS server stores limited Health Check data. The primary role of Health Check is to look for anomalies and generate RAS when abnormal readings are detected.

The polling intervals can be changed without affecting users.

The monitoring can be stopped by issuing the `stop_hw_polling` command from `bg_console` without impacting users on the system.

12.5.3 Data collection and RAS

This section describes the data that is collected by MMCS server during environmental polling, the RAS messages that might be generated, and if applicable, the database table that the collected data is stored in.

Service card monitor

The service card monitors the voltages of the following persistent power rails:

- ▶ Voltage on 1.2 V persistent power rail
- ▶ Voltage on 1.25 V persistent power rail
- ▶ Voltage on 1.5 V persistent power rail
- ▶ Voltage on 2.5 V persistent power rail
- ▶ Voltage on 3.3 V persistent power rail
- ▶ Voltage on 5.0 V persistent power rail

Collected data is stored in the `BGQServiceCardEnvironment` table.

If the service card monitor reports an error contacting a service card, a RAS message 00061002 is generated.

Node board monitor

Monitors the following readings on the node boards:

- ▶ Voltages:
 - 0.8v rail
 - 1.4v rail
 - 2.5v rail
 - 3.3v rail
 - 12.0v rail (persistent)
 - 1.5v rail
 - 0.9v rail
 - 1.0v rail
 - 3.3v rail (persistent)
 - V12A rail
 - V12B rail
 - 1.8v rail
 - 2.5v rail (persistent)
 - 1.2v rail (persistent)
 - 1.8v rail (persistent)

Collected data is stored in the `BGQNodeCardEnvironment` table.

- ▶ Temperature sensors:
 - On-board temperature sensor 0
 - On-board temperature sensor 1

Collected data is stored in the `BGQNodeEnvironment` table.

- ▶ ASIC temperature

Collected data is stored in the `BGQLinkChipEnvironment` table.

If the node board monitor reports an error contacting a node board, a RAS message 00061001 is generated.

I/O board monitor

Monitors the following readings on the I/O node board:

- ▶ Voltages:
 - Voltage on 0.8v power rail
 - Voltage on 1.4v power rail
 - Voltage on 2.5v power rail
 - Voltage on 3.3v power rail
 - Voltage on 12.0v power rail
 - Voltage on 1.5v power rail
 - Voltage on 0.9v power rail
 - Voltage on 1.0v power rail
 - Voltage on 12.0v power rail (persistent)
 - Voltage on 3.3v power rail (persistent)
 - Voltage on 1.2v power rail
 - Voltage on 1.8v power rail
 - Voltage on 1.2v power rail (persistent)
 - Voltage on 1.5v power rail (persistent)
 - Voltage on 1.8v power rail (persistent)
 - Voltage on 2.5v power rail (persistent)

Collected data is stored in the IOCardEnvironment table.

- ▶ Temperature sensors:

- On-board temperature sensor

Collected data is stored in the BGQNodeEnvironment table.

- ▶ Fan speed (in RPMs)

Collected data is stored in the BGQFanEnvironment table.

- ▶ ASIC temperature

Collected data is stored in the BGQLinkChipEnvironment table.

If the I/O board monitor reports an error contacting an I/O board, a RAS message 00061004 is generated.

Performance data monitor

Performance data is pulled from the *Low-Level Control System* (LLCS), which contains primarily boot time performance counters.

Collected data is stored in the BGQComponentPerf table.

MMCS server: The MMCS server tracks how much time it takes to gather environmentals. Turning off the gathering of performance data not only stops the MMCS server from gathering performance data from LLCS but also stops MMCS server from reporting how long it took to collect any environmental data.

Coolant monitor

Monitors the following readings for coolant:

- ▶ Inlet coolant flow rate
- ▶ Outlet coolant flow rate

- ▶ Differential coolant pressure
- ▶ Coolant supply temperature
- ▶ Coolant return temperature
- ▶ Dew-point temperature
- ▶ Ambient-air temperature
- ▶ Ambient-air relative humidity (in percentage)
- ▶ System power being consumed by this rack (calculated by using flow rate and coolant temperature difference)

Collected data is stored in the BGQCoolantEnvironment table.

If the coolant monitor cannot be contacted, a RAS message 00061005 is generated.

Bulk power module (BPM) monitor

Monitors the following voltage and current on the BPMs:

- ▶ Input voltage
- ▶ Input current
- ▶ Output voltage
- ▶ Output current

Collected data is stored in the BGQBulkPowerEnvironment table.

If the BPM monitor reports an error contacting a BPM, a RAS message 00061003 is generated.

Optical module monitor

Monitors the following optical components:

- ▶ Optical channel
- ▶ Optical power

Collected data is stored in the BGQOpticalEnvironment table.

If the optical module monitor reports an error contacting a node board, a RAS message 00061001 is generated.

If the optical module monitor reports an error contacting an I/O board, a RAS message 00061004 is generated.

Health Check

The Health Check monitors hardware and generates RAS for abnormal conditions. It also monitors service card on-board temperature and stores collected data in the BGQServiceCardTemp table.

Note: The service card on-board temperature is the only reading stored by Health Check. The reason for this is because service card temperatures need to be collected on a shorter interval than the service card voltages. It was decided to process the service temperatures at the same interval as the service card Health Check, which runs more frequently than the service card monitor. The default interval for service cards is 30 minutes and the Health Check runs on a 5 minute interval.

12.5.4 Location-specific monitoring

There are commands in `bg_console` to turn on and off polling for a specific location of a service card, I/O board, node board, or bulk power module. The commands are `start_hw_polling` and `stop_hw_polling`. The `start_hw_polling` command does the same processing as the monitor details listed previously, but allows a different polling interval for a single location than the interval used for other locations of that type. The `start_hw_polling` command augments the normal polling with an additional, location-specific polling interval. This is useful when trying to diagnose problems on a certain piece of hardware, and it is necessary to poll the hardware more often.

The command `list_hw_polling` shows any location-specific polling that is already in effect.

12.5.5 RAS events

The following RAS events are generated by the environmental monitor. They are written to the `BGQEventLog` table.

Service cards:

- ▶ RAS 00061002 if service card cannot be contacted
- ▶ RAS 00061006 if service card returns leak detector fault
- ▶ RAS 00061007 if service card clock frequency outside 1520 - 1680
- ▶ RAS 00061008 if service card overtemp

Node boards:

- ▶ RAS 00061001 if node board cannot be contacted
- ▶ RAS 00061010 if node board clock frequency outside 1520 - 1680
- ▶ RAS 00061011 if alert or pgood warning for blinks
- ▶ RAS 00061011 if alert or pgood warning for computes
- ▶ RAS 00061011 if alert for optical modules
- ▶ RAS 00061011 if non-zero status for DCA
- ▶ RAS 00061011 if non-zero status for VTM domains:
 - DCAs on node boards:
 - RAS 0006100C if non-zero status for DCA
 - RAS 0006100C if error flag on for DCA
 - RAS 0006100C if invalid domain value returned for DCA
 - RAS 0006100C if DCA voltage outside of range (see Table 12-1)
 - RAS 0006100C if DCA current outside of range (see Table 12-1)

Table 12-1 Voltage and current ranges for node boards

Domain	Minimum voltage	Maximum voltage	Maximum current readback mA
1	.7	1.1	57280
2	1.2	1.5	26080
3	2.25	2.75	2115
4	3.3	3.6	2203
5	n/a	n/a	n/a
6	1.45	1.55	4771
7	.7	1.2	4771

Domain	Minimum voltage	Maximum voltage	Maximum current readback mA
8	.9	1.1	1740
9	11.4	14.4	n/a

- Optical Modules on node boards:
 - RAS 0006100E if error flag on for optical module
 - RAS 0006100E if loss of signal flag on for optical module
 - RAS 0006100E if module fault flag on for optical module
 - RAS 0006100E if temperature flag on for optical module
 - RAS 0006100E if voltage flag on for optical module
 - RAS 0006100E if bias current flag on for optical module
 - RAS 0006100E if power flag on for optical module
 - RAS 0006100E if interrupt flag for module status is on, but none of the above error flags are on (shouldn't happen)

I/O boards:

- ▶ RAS 00061004 if I/O board cannot be contacted
- ▶ RAS 00061009 if I/O board clock frequency outside 1520 - 1680
- ▶ RAS 0006100A if temp critical or temp warning for blinks
- ▶ RAS 0006100A if alert or pgood warning for blinks
- ▶ RAS 0006100A if alert or pgood warning for computes
- ▶ RAS 0006100A if alert for optical modules
- ▶ RAS 0006100A if non-zero status for DCA
- ▶ RAS 0006100A if non-zero status for VTM domains:
 - DCAs on I/O boards:
 - RAS 0006100B if non-zero status for DCA
 - RAS 0006100B if error flag on for DCA
 - RAS 0006100B if invalid domain value returned for DCA
 - RAS 0006100B if DCA voltage outside of range (see Table 12-2)
 - RAS 0006100B if DCA current outside of range (see Table 12-2)

Table 12-2 Voltage and current ranges for I/O boards

Domain	Minimum voltage	Maximum voltage	Maximum current readback mA
1	.7	1.1	13480
2	1.2	1.5	6490
3	2.25	2.75	n/a
4	3.3	3.6	n/a
5	11.0	13.0	6370
6	1.45	1.55	n/a
7	.7	1.2	n/a
8	.9	1.1	n/a
9	11.0	13.0	n/a

- Optical Modules on I/O boards:

- RAS 0006100D if error flag on for optical module
- RAS 0006100D if loss of signal flag on for optical module
- RAS 0006100D if module fault flag on for optical module
- RAS 0006100D if temperature flag on for optical module
- RAS 0006100D if voltage flag on for optical module
- RAS 0006100D if bias current flag on for optical module
- RAS 0006100D if power flag on for optical module
- RAS 0006100D if interrupt flag for module status is on, but none of the above error flags are on

Bulk power modules:

- ▶ RAS 00061003 if bulk power module has error flag set
- ▶ RAS 0006100F if bulk power module has non-zero general status word
- ▶ RAS 0006100F if bulk power module has non-zero output voltage status
- ▶ RAS 0006100F if bulk power module has non-zero output current status
- ▶ RAS 0006100F if bulk power module has non-zero input power status
- ▶ RAS 0006100F if bulk power module has non-zero temperature status
- ▶ RAS 0006100F if bulk power module has non-zero CML status (comm., logic and memory)
- ▶ RAS 0006100F if bulk power module has non-zero 5 volt output status
- ▶ RAS 0006100F if bulk power module has non-zero fan status
- ▶ RAS 0006100F if bulk power module has 5v output voltage not in 4.0-5.6 range
- ▶ RAS 0006100F if bulk power module has 5v output current not in 0.0-2.2 range
- ▶ RAS 0006100F if bulk power module has any temperature modules over 70
- ▶ RAS 0006100F if bulk power module has any fans outside 0-25000 rpm range
- ▶ RAS 00061012 for affected node boards under certain bulk power failure conditions:
 - 8 per BPE in compute rack
- ▶ RAS 00061013 for affected I/O boards under certain bulk power failure conditions:
 - 1 per BPE in compute rack or all I/O boards in I/O rack



Machine controller server

This chapter describes the machine controller (MC) server, which provides an interface to the *Low-Level Control System* (LLCS).

MC server is also referred to by its alternate name, mc_server, in this chapter.

13.1 Machine controller server

The MC server is a wrapper around the machine controller, which is a library of functions that provides *Low-Level Control System* (LLCS) access to the Blue Gene/Q hardware. MC server wraps this library to provide a server and ports to which to connect. MC server handles arbitrations and controls access to the hardware to prevent simultaneous conflicting operations. It also provides target sets, which give the client a way to refer to a collection of hardware with a single name or handle. MC server also provides some event handling. Clients can register to receive RAS and console events for a particular location or for all events.

Figure 13-1 illustrates how MC server, shown in Figure 13-1, provides a layer of arbitration between the hardware and higher level clients. Examples of clients are MMCS, diagnostics, and service actions.

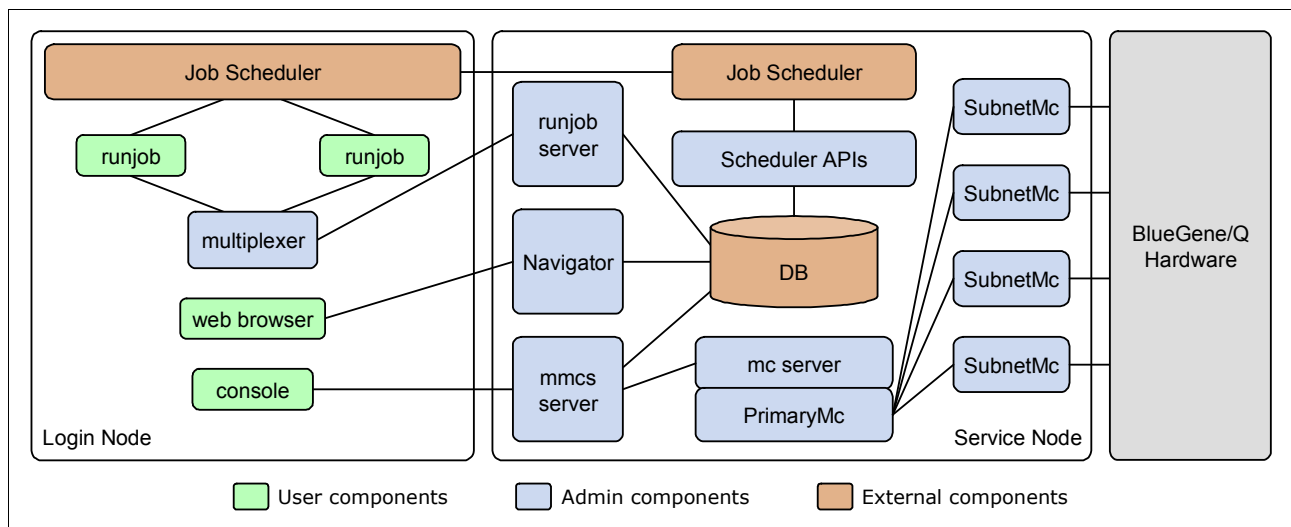


Figure 13-1 Control System architecture

13.2 Starting and stopping MC server

The MC server executable is `/bgsys/drivers/ppcfloor/control/sbin/mc_server_64`. Starting and stopping the MC server is controlled by BGMaster; however, it can be done interactively with the `master_stop` or `master_start` commands described in section 11.2, “Running bgmaster_server” on page 130

13.3 Configuration

The MC server can be configured by setting options in the Blue Gene/Q configuration file or by setting command-line options when the server starts.

13.3.1 Configuration file options

Configuration options used by the MC server are in the [machinecontroller] section of the Blue Gene/Q configuration file (/bgsys/local/etc/bg.properties). Configurable options for the MC server include (default values are shown):

- ▶ `clientPort = 1206`
The TCP/IP port that MC server will listen on.
- ▶ `hostName = 127.0.0.1`
The IP address or host name that external applications use to communicate with MC server. IP addresses containing colons must be enclosed in []. Link local ipv6 addresses must include the interface name appended after a % character.
- ▶ `bringup = true`
Bringup the hardware when MC server starts. Valid values are true or false.
- ▶ `spillFileDir =`
Fully qualified path to the RAS spill file directory. If not specified, no spill file is used.
- ▶ `alteraPath = /bgsys/drivers/ppcfloor/ctrlnet/bin/icon/fpgaImages`
Fully qualified path to the images for the iCon/Palominos (FPGAs).
- ▶ `CardThatTalksToMasterClockCard = R00-M0-S`
The location of the card that communicates with/controls the master clock card.
- ▶ `kernel_shutdown_timeout = 60`
Number of seconds within which the compute and I/O kernels must complete shutdown.
- ▶ `io_link_shutdown_timeout = 15`
Number of seconds to complete I/O link shutdown.

13.3.2 Command-line options

The MC server (`mc_server_64` executable) accepts the following options on the command line:

- `--properties arg` This option provides the path to an alternate Blue Gene/Q configuration file. The default file, /bgsys/local/etc/bg.properties or the location specified by BG_PROPERTIES_FILE environment variable if defined is used if the option is not specified.
- `-h [--help]` Prints the help text.

13.4 The `mc_server_log_level` command

The `mc_server_log_level` command displays and changes the logging levels of the `mc_server`. With no logger arguments, the logger names and their levels for the `mc_server` are displayed. To set logger levels, specify their name and a level as positional arguments. The valid logging levels are the same values that are supported by the `--verbose` argument. The syntax of the command is:

```
mc_server_log_level [OPTIONS ... ] logger=level [... logger=level]
```

The accepted arguments for the `mc_server_log_level` command are:

- host** Defines the server host to connect to. Enclose IPv6 addresses in square brackets. For link-local addresses, append the interface to the address after a '%' character. The following values are examples values:
172.16.3.1
[::1]
- port** Defines the server port to connect to.
- verbose** Sets the logging configuration. Setting these values overrides the logging configuration in the bg.properties file. The --verbose option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level; OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the ibm.utility logger to the *debug* value, the following command can be used:
mc_server_log_level --verbose=DEBUG --verbose=ibm.utility=DEBUG
- properties** This option provides the path to an alternate Blue Gene/Q configuration file. The default file, /bgsys/local/etc/bg.properties or the location specified by the BG_PROPERTIES_FILE environment variable, if defined, is used if the option is not specified.
- version** Displays version information
- help** Prints the help text.



The runjob server and runjob mux

The umbrella term for the Blue Gene/Q job submission architecture is *runjob*. This chapter provides information about the runjob server, which facilitates remote job launch for the Blue Gene/Q Control System and the runjob multiplexer (mux), which communicates with both the runjob server and runjob client to relay requests and responses.

For information about submitting jobs, job states, and job-related commands, see Chapter 6. “Submitting jobs” on page 75.

14.1 The runjob architecture

The runjob architecture, shown in Figure 14-1, consists of three primary components: the runjob server, the multiplexer (runjob mux), and runjob clients:

- ▶ The role of the runjob server is to insert, update, and remove job entries from the database. It also communicates with daemons on the I/O nodes to start and stop jobs. The runjob server communicates with multiplexer to receive and respond to client requests.
- ▶ The runjob multiplexer (mux) communicates with both the runjob server and client to relay requests and responses. The mux acts like a proxy to funnel multiple job requests into a single connection to the server. The existence of multiple multiplexers facilitates failover. There is plug-in functionality to interact with job schedulers.
- ▶ The runjob client acts as an interface for job schedulers. The client communicates with multiplexer to send requests to the runjob server.

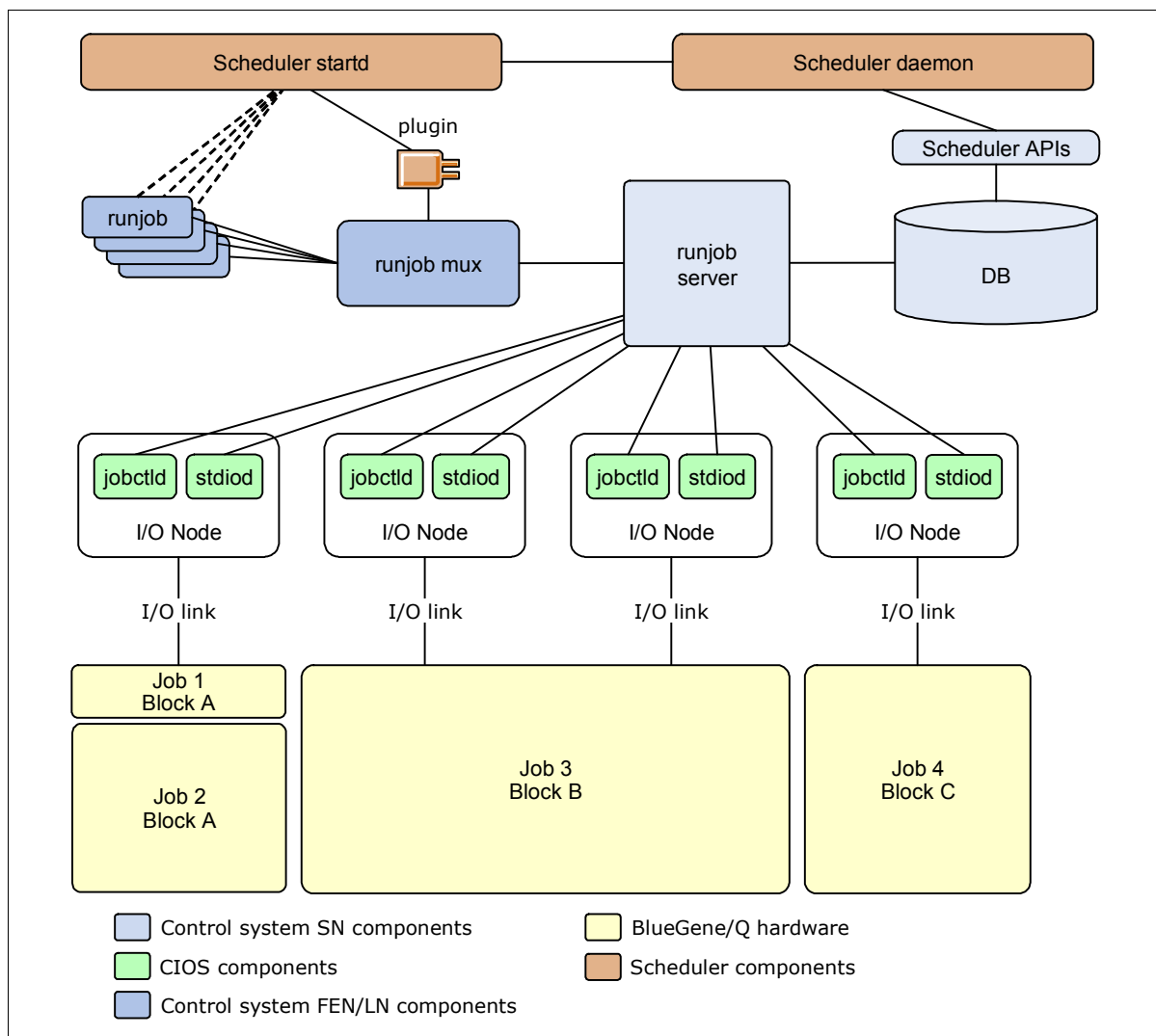


Figure 14-1 runjob components

14.2 The runjob_server

The runjob_server receives job requests from the runjob_mux, and then inserts, updates, and removes job entries from the database. It also communicates with daemons on the I/O nodes to start and stop jobs and tool daemons for jobs.

14.2.1 Configuration file options

Configuration options used by runjob_server are in the [runjob.server] section of the Blue Gene/Q configuration file (/bgsys/local/etc/bg.properties). Configurable options include:

- ▶ thread_pool_size = auto

The number of threads to run in the server. The allowed values are integers greater than 0 or auto. Specifying auto, which is the default, allows the runjob_server to pick the number of threads to start.

- ▶ mux_listen_ports = localhost:25510

command_listen_ports = localhost:24510

The ports that the runjob_server will listen for command and multiplexer connections on. The format is a comma-separated list of host:port pairs. The *host* is an IP address or host name. IP addresses containing colons must be enclosed in square brackets []. The *port* is a port number or service name. The host and port are optional. If the host is not specified, the server will listen on any interface. If the port is not specified, the default port is used (25510 and 24510).

- ▶ io_connection_interval_increment = 5

The number of seconds to increment a progressive timeout when retrying connection attempts to the common I/O services (CIOS) daemons. This value can be updated by the **runjob_server_refresh_config** command.

- ▶ io_connection_interval_max = 120

Maximum number of seconds to wait between connection attempts to the CIOS daemons. This value can be updated by the **runjob_server_refresh_config** command.

- ▶ performance_counter_interval = 30

Number of seconds to wait between inserting performance counter statistics into the database for persistent storage. If this value is not specified the default value of 30 seconds is used. This value can be updated by the **runjob_server_refresh_config** command.

- ▶ connection_pool_size = 20

The maximum number of connections held in the runjob_server database connection pool. If not specified, the maximum is 20.

- ▶ extra_connection_parameters = Debug=True;

Extra parameters to pass on the connection. Format is NAME=VALUE pairs separated by a semicolon (;). Optional, default is no extra parameters.

- ▶ control_action_heartbeat = 60

The number of seconds to wait between checking for progress of a job that was terminated because of a RAS event with a Control Action. If not specified, this value is 60. This value can be updated by the **runjob_server_refresh_config** command.

The runjob_server_refresh_config command

Issuing the `runjob_server_refresh_config` command refreshes the `runjob_server` configuration. The syntax of the `runjob_server_refresh_config` command is:

```
runjob_server_refresh_config [file] [OPTIONS ... ]
```

The accepted arguments for the `runjob_server_refresh_config` command are:

--file	The configuration file to reload. This parameter is optional. If not specified, the current configuration file is reloaded. This parameter can be specified as the first positional argument.
--host	Defines the server host and port to connect to. Enclose IPv6 addresses in square brackets. For link-local addresses, append the interface to the address after a '%' character. Specify the port after the interface using a colon. Example values follow: 127.0.0.1 172.16.3.1:12345 [::1] [fe80::214:5eff:fre9c:52ce%eth0] [::1]:24555
--wait-for-server	Keeps trying to connect to the server if it is unavailable.
--verbose arg	Sets the logging configuration. Setting these values overrides the logging configuration in the <code>bg.properties</code> file. The <code>--verbose</code> option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the <code>ibm.utility</code> logger to the <code>debug</code> value, the following command can be used: runjob_server_refresh_config --verbose=DEBUG --verbose=ibm.utility=DEBUG
--properties	This option provides the path to an alternate Blue Gene/Q configuration file. The default file, <code>/bgsys/local/etc/bg.properties</code> or the location specified by <code>BG_PROPERTIES_FILE</code> environment variable if defined is used if the option is not specified.
--version	Displays version information
--help	Prints the help text.

14.2.2 Command-line options

Starting and stopping the `runjob_server` is controlled by `BGmaster`; however, it can be done interactively with the `master_stop` or `master_start` commands described in section 11.2 “Running `bgmaster_server`” on page 130. The active server’s behavior can be modified with the `runjob_server` command by using the following syntax:

```
runjob_server [OPTIONS ... ]
```

The accepted arguments for the `runjob_server` command are:

--sim= arg	Enables or disables job simulation. The possible values are true or false. This overrides the <code>job_sim</code> option in the <code>[runjob.server]</code> section of the <code>bg.properties</code> configuration file.
-------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Note: The `--sim` argument must always be set to “false” in a production environment. A Control System simulator environment is a special environment for testing without actual Blue Gene/Q hardware. See Appendix B, “Control System simulator” on page 247 for more details.

--mux-listen-port arg Defines the address and port to bind to for multiplexer connections. This overrides the `mux_listen_ports` option in the `[runjob.server]` section of the `bg.properties` configuration file. Enclose IPv6 addresses in square brackets. For link-local addresses append the interface to the address after a ‘%’ character. Optionally, specify the port after the address using a colon. Example values follow:

127.0.0.1

172.16.3.1:12345

[::1]

[fe80::214:5eff:fre9c:52ce%eth0]

[::1]:24555

The default port is 25510.

--command-listen-port arg

Defines the address and port to bind to for command connections. This overrides the `command_listen_ports` option in the `[runjob.server]` section of the `bg.properties` configuration file. Enclose IPv6 addresses in square brackets. For link-local addresses append the interface to the address after a ‘%’ character. Optionally, specify the port after the address using a colon. Example values follow:

127.0.0.1

172.16.3.1:12345

[::1]

[fe80::214:5eff:fre9c:52ce%eth0]

[::1]:24555

The default port is 24510.

--properties arg

This option provides the path to an alternate Blue Gene/Q configuration file. The default file, `/bgsys/local/etc/bg.properties` or the location specified by `BG_PROPERTIES_FILE` environment variable, if defined, is used if the option is not specified.

--verbose arg

Sets the logging configuration. Setting these values overrides the logging configuration in the `bg.properties` file. The `--verbose` option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the `ibm.utility` logger to the `debug` value, the following command can be used:

runjob_server --verbose=DEBUG --verbose=ibm.utility=DEBUG

-v [--version]

Displays version information.

-h [--help]

Command help text.

14.3 The runjob_mux

The runjob_mux receives multiple client requests and communicates them using a single connection to the runjob_server.

14.3.1 Configuration file options

The following options are configurable in the [runjob.mux] section of the /bgsys/local/etc/bg.properties file:

- ▶ `thread_pool_size = auto`
Sets the number of threads to run in the runjob_mux process. The allowed values are integers greater than 0 or auto. Setting the size to auto means the runjob_mux will pick the number of threads to start. The default value for this option is auto.
- ▶ `plugin =`
The *plugin* value provides the fully qualified path to the plug-in used for communicating with a job scheduler. This value can be updated by the **runjob_mux_refresh_config** command on the front end node where a runjob_mux process runs. Information about the Scheduler APIs can be accessed on the Knowledge Center tab in Blue Gene Navigator.
- ▶ `plugin_flags =`
Hexadecimal flags to pass as the second argument to dlopen. If not specified, the default value is 0x0001 (RTLD_LAZY). See the dlopen(3) man page for more information.
- ▶ `local_socket = runjob_mux`
Name of the local AF_UNIX socket to use when listening for runjob connections. It is placed in the abstract namespace, see man UNIX(7) for more information.
- ▶ `host = localhost:25510`
The host and port to connect to. The runjob_server should be listening on the port. The format is a comma-separated list of host:port pairs. Host is an IP address or host name. IP addresses containing colons must be enclosed in square brackets. Link local IPv6 addresses must include the interface name appended after a '%' character. Port is a port number or service name. The host and port are optional. If the host is not specified, the client will connect to localhost. If the port is not specified the default port is used (25510).
- ▶ `command_listen_ports = localhost:26510`
The ports that the runjob_mux will listen for connections on. The format is a comma-separated list of host:port pairs. Host is an IP address or host name. IP addresses containing colons must be enclosed in square brackets. Link local IPv6 addresses must include the interface name appended after a '%' character. Port is a port number or service name. The host and port are optional. If the host is not specified, the runjob_mux process will listen on any interface. If the port is not specified, the default port is used (26510).
- ▶ `performance_counter_interval = 30`
Number of seconds to wait between sending performance counter statistics to the runjob_server for persistent storage in the database. Default is 15 seconds if this value is not specified. This value can be updated by the **runjob_mux_refresh_config** command on the front end node where a runjob_mux process runs.
- ▶ `server_connection_interval = 10`
Number of seconds to wait between connection attempts to the runjob_server when it is unavailable. Default value is 10 seconds if not specified here. This value can be updated

by the `runjob_mux_refresh_config` command on the front end node where a `runjob_mux` process runs.

- ▶ `client_output_buffer_size = 512`

Maximum number of kilobytes of stdout or stderr messages to queue up per client. Messages received after this value is reached are dropped. Default value is 512 kilobytes if not specified here. This value can be updated by the `runjob_mux_refresh_config` command on the front end node where a `runjob_mux` process runs.

The `runjob_mux_refresh_config` command

Issuing the `runjob_mux_refresh_config` command refreshes the `runjob_mux` configuration. The syntax of the `runjob_mux_refresh_config` command is:

```
runjob_mux_refresh_config [OPTIONS ... ]
```

The accepted arguments for the `runjob_mux_refresh_config` command are:

- | | |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| --host | Defines the server host and port to connect to. Enclose IPv6 addresses in square brackets. For link-local addresses, append the interface to the address after a '%' character. Specify the port after the interface using a colon. Example values are:

127.0.0.1
172.16.3.1:12345
[::1]
[fe80::214:5eff:fre9c:52ce%eth0]
[::1]:24555 |
| --wait-for-server | Keep trying to connect to the server if it is unavailable. |
| --verbose arg | Sets the logging configuration. Setting these values overrides the logging configuration in the <code>bg.properties</code> file. The <code>--verbose</code> option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the <code>ibm.utility</code> logger to the <code>debug</code> value, the following command can be used:
runjob_mux_refresh_config --verbose=DEBUG
--verbose=ibm.utility=DEBUG |
| --properties | This option provides the path to an alternate Blue Gene/Q configuration file. The default file, <code>/bgsys/local/etc/bg.properties</code> , or the location specified by the <code>BG_PROPERTIES_FILE</code> environment variable, if defined, is used if the option is not specified. |
| --version | Displays version information |
| --help | Prints the help text. |

14.3.2 Command-line options

Typically, the runjob multiplexer is controlled by BGmaster; however, the multiplexer can be started or stopped interactively using the **master_start** or **master_stop** command described in section 11.2 “Running bgmaster_server” on page 130. Changing the running server’s options can be done with the **runjob_mux** command located in the `/bgsys/drivers/ppcfloor/hlcs/sbin` directory. The syntax of the **runjob_mux** command is:

```
runjob_mux [OPTIONS ... ]
```

The accepted arguments for the **runjob_mux** command are:

- host** Defines the runjob_server host and port to connect to. This overrides the host option in the [runjob.mux] section of the bg.properties configuration file. Enclose IPv6 addresses in square brackets. For link-local addresses, append the interface to the address after a ‘%’ character. Specify the port after the interface using a colon. Example values are:
 - 127.0.0.1
 - 172.16.3.1:12345
 - :::1
 - [fe80::214:5eff:fre9c:52ce%eth0]
 - :::1:24555
- command-listen-port** Defines the address and port to bind to for command connections. This overrides the command_listen_ports option in the [runjob.mux] section of the bg.properties configuration file. Enclose IPv6 addresses in square brackets. For link-local addresses, append the interface to the address after a ‘%’ character. Optionally, specify the port after the address using a colon. Example values are:
 - 127.0.0.1
 - 172.16.3.1:12345
 - :::1
 - [fe80::214:5eff:fre9c:52ce%eth0]
 - :::1:24555
- socket** Name of the runjob_mux listen socket, maximum length is 107 characters. This is stored in the abstract namespace rather than on the file system. This overrides the local_socket option in the [runjob.mux] section of the bg.properties configuration file.
- verbose arg** Sets the logging configuration. Setting these values overrides the logging configuration in the bg.properties file. The --verbose option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the ibm.utility logger to the *debug* value, the following command can be used:


```
runjob_mux --verbose=DEBUG --verbose=ibm.utility=DEBUG
```
- properties** This option provides the path to an alternate Blue Gene/Q configuration file. The default file, `/bgsys/local/etc/bg.properties`, or the location specified by the `BG_PROPERTIES_FILE` environment variable, if defined, is used if the option is not specified.

--version	Displays version information.
--help	Prints the help text.

14.4 Utilities

Utilities are provided to get the status of the runjob server and runjob mux and change the logging configuration of both the server and the mux. The following utilities are provided:

- ▶ `runjob_server_status`
- ▶ `runjob_server_log_level`
- ▶ `runjob_mux_status`
- ▶ `runjob_mux_log_level`

14.4.1 Configuration options

The `runjob_server` and `runjob_mux` utilities use the `[runjob.server.commands]` and `[runjob.mux.commands]` subsections of the `bg.properties` file to direct their queries.

In the `[runjob.server.commands]` subsection, the `host` key identifies the host name and port that the `runjob_server` is listening on.

```
host = localhost:24510
```

Similarly, in the `[runjob.mux.commands]` subsection, the `host` key identifies the host name and port that the `runjob_mux` is listening on.

```
host = localhost:26510
```

In both cases, the format of the `host` value is a comma-separated list of `host:port` pairs. Host can be an IP address or host name. IP addresses containing colons must be enclosed in square brackets. Link local IPv6 addresses must include the interface name appended after a `'%'` character. The port specified is a port number or service name. The host and port are optional. If the host is not specified, the client connects to `localhost`. If the port is not specified, the default port is used. This port is 24510 for the `runjob_server` and 26510 for the `runjob_mux`.

14.4.2 The `runjob_server_status` command

The `runjob_server_status` command displays the status of the `runjob_server`. Example 14-1 shows the output of the command. Included in the output is information about CIOS, performance counters, server and mux connections, jobs, blocks and I/O links.

Example 14-1 runjob_server_status output

```
[sbin]$ runjob_server_status
BG/Q runjob server
driver: bgq
revision: 53064
configured from: /bgsys/local/etc/bg.properties
load: 21 microseconds
realtime block notifications

CIOS protocol
8 job control
4 standard I/O
```

```

Performance Counters
8 jobs
0 miscellaneous

3 mux connections
2 command connections

4 jobs

34 blocks

176 I/O links
88 jobctl links up 0 links down
88 stdio links up 0 links down

```

The accepted arguments for the **runjob_server_status** command are:

--host	Defines the server host and port to connect to. Enclose IPv6 addresses in square brackets. For link-local addresses, append the interface to the address after a ‘%’ character. Specify the port after the interface using a colon. Example values are: 127.0.0.1 172.16.3.1:12345 [::1] [fe80::214:5eff:fre9c:52ce%eth0] [::1]:24555
--wait-for-server	Keeps trying to connect to the server if it is unavailable.
--verbose arg	Sets the logging configuration. Setting these values overrides the logging configuration in the bg.properties file. The --verbose option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the ibm.utility logger to the <i>debug</i> value, the following command can be used: runjob_server_status --verbose=DEBUG --verbose=ibm.utility=DEBUG
--properties	This option provides the path to an alternate Blue Gene/Q configuration file. The default file, /bgsys/local/etc/bg.properties or the location specified by BG_PROPERTIES_FILE environment variable if defined is used if the option is not specified.
--version	Displays version information
--help	Prints the help text.

14.4.3 The runjob_server_log_level command

The **runjob_server_log_level** command displays and changes the logging levels of the runjob_server. With no logger arguments, the logger names and their levels of the runjob_server are displayed. To set logger levels, specify their name and a level as positional arguments. The valid logging levels are the same values that are supported by the --verbose argument. The syntax of the **runjob_server_log_level** command is:

runjob_server_log_level [OPTIONS ...] logger=level [... logger=level]

The accepted arguments for the **runjob_server_log_level** command are:

- host** Defines the server host and port to connect to. Enclose IPv6 addresses in square brackets. For link-local addresses, append the interface to the address after a '%' character. Specify the port after the interface using a colon. Example values are:
 127.0.0.1
 172.16.3.1:12345
 [::1]
 [fe80::214:5eff:fre9c:52ce%eth0]
 [::1]:24555
- wait-for-server** Keeps trying to connect to the server if it is unavailable.
- verbose** Sets the logging configuration. Setting these values overrides the logging configuration in the bg.properties file. The --verbose option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the ibm.utility logger to the *debug* value, the following command can be used:
**runjob_server_log_level --verbose=DEBUG
 --verbose=ibm.utility=DEBUG**
- properties** This option provides the path to an alternate Blue Gene/Q configuration file. The default file, /bgsys/local/etc/bg.properties or the location specified by BG_PROPERTIES_FILE environment variable if defined is used if the option is not specified.
- version** Displays version information
- help** Prints the help text.

Example 14-2 shows the output from the **runjob_server_log_level** command using no arguments, which displays all loggers and their levels.

Example 14-2 runjob_server_log_level command output

```
$ runjob_server_log_level
ibm                               INFO
ibm.agent                         INFO
ibm.bgqconfig                     INFO
ibm.bgsched                       INFO
ibm.bgws                          INFO
ibm.boot                          DEBUG
ibm.cios                          WARN
ibm.database                      INFO
ibm.icon                          DEBUG
ibm.master                        INFO
ibm.mc                             DEBUG
ibm.mc.MCServer                   DEBUG
ibm.mc.MCServerLockNode           DEBUG
ibm.mc.MCServerThread             DEBUG
ibm.mmcs                          INFO
ibm.ras                           WARN
ibm.realtime                      INFO
ibm.runjob                        WARN
```

ibm.runjob.mux	DEBUG
ibm.runjob.server	DEBUG
ibm.security	INFO
ibm.utility	INFO
ibm.utility.cxxsockets	WARN

To change the ibm.utility logger to TRACE level, use the following command:

```
runjob_server_log_level1 ibm.utility=TRACE
```

Multiple logger levels can be changed using a single command. The following command updates the ibm.runjob logger to WARN level and the ibm.database logger to DEBUG level.

```
runjob_server_log_level1 ibm.runjob=WARN ibm.database=DEBUG
```

14.4.4 The runjob_mux_status command

The `runjob_mux_status` command displays the current status of the `runjob_mux`. Example 14-3 shows the output of the `runjob_mux_status` command.

Example 14-3 runjob_mux_status output

```
BG/Q runjob multiplexer
driver: bgq
revision: 50784
configured from: /bgsys/local/etc/bg.properties
load: 47 microseconds

runjob server: localhost

scheduler plugin: loaded from: /usr/lib64/lib11runjob_mux.so
bgsched version: 1.0.0

1 connection
Host      Port  Type
localhost 58813 command

5 clients
  pid Client Username      Job Queue Size Max Size Dropped Messages
  29184 2214 dougjpv1 174029      0    524288      0
  7622 2282 kmsftr 174119      0    524288      0
  1815 1843 smithjq 173706      0    524288      0
  12222 2339 dougjpv1 174181      0    524288      0
  26100 2340 jpvldi11 174183      0    524288      0
```

The accepted arguments for the `runjob_mux_status` command are:

--host	Defines the server host and port to connect to. Enclose IPv6 addresses in square brackets. For link-local addresses, append the interface to the address after a '%' character. Specify the port after the interface using a colon. Example values are:
	127.0.0.1
	172.16.3.1:12345
	[::1]
	[fe80::214:5eff:fre9c:52ce%eth0]
	[::1]:24555

--wait-for-server	Keep trying to connect to the server if it is unavailable.
--verbose arg	Sets the logging configuration. Setting these values overrides the logging configuration in the <code>bg.properties</code> file. The <code>--verbose</code> option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the <code>ibm.utility</code> logger to the <code>debug</code> value, the following command can be used: runjob_mux_status --verbose=DEBUG --verbose=ibm.utility=DEBUG
--properties	This option provides the path to an alternate Blue Gene/Q configuration file. The default file, <code>/bgsys/local/etc/bg.properties</code> or the location specified by <code>BG_PROPERTIES_FILE</code> environment variable if defined is used if the option is not specified.
--version	Displays version information
--help	Prints the help text.

14.4.5 The `runjob_mux_log_level` command

The `runjob_mux_log_level` command is used to display and change the logging levels for the `runjob_mux`. When no logger arguments are submitted the current logger names and their levels of the `runjob_mux` are displayed. To set logger levels, specify the name of the logger and a level as positional arguments. The valid logging levels are the same values that are supported by the `verbose` argument.

The accepted arguments for the `runjob_mux_log_level` command are:

--host	Defines the server host and port to connect to. Enclose IPv6 addresses in square brackets. For link-local addresses, append the interface to the address after a '%' character. Specify the port after the interface using a colon. Example values follow: 127.0.0.1 172.16.3.1:12345 [::1] [::1]:24555
--wait-for-server	Keep trying to connect to the server if it is unavailable.
--verbose arg	Sets the logging configuration. Setting these values overrides the logging configuration in the <code>bg.properties</code> file. The <code>--verbose</code> option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the <code>ibm.utility</code> logger to the <code>debug</code> value, the following command can be used: runjob_mux_log_level --verbose=DEBUG --verbose=ibm.utility=DEBUG
--properties	This option provides the path to an alternate Blue Gene/Q configuration file. The default file, <code>/bgsys/local/etc/bg.properties</code> , or the location specified by <code>BG_PROPERTIES_FILE</code> environment variable, if defined, is used if the option is not specified.
--version	Displays version information
--help	Prints the help text.



The Blue Gene Web Services server

The Blue Gene Web Services (BGWS) provides a RESTful web service for the Blue Gene/Q system. As a web service, the BGWS can be used by any web client, either from a web application running in a browser, such as the Blue Gene Navigator, or other programs, such as a command-line utility like `list_jobs`.

The BGWS are documented in the online documentation shipped with Blue Gene/Q in the `/bgsys/drivers/ppcfloor/bgws/doc/html` directory. You can also display the BGWS documentation from the Knowledge Center in the Navigator.

15.1 Configuring and running the BGWS server

Administrators must follow these instructions to configure and run the BGWS server:

1. Download the prerequisite packages. It is recommended to configure the Apache server to use SSL, which is enabled by installing the `mod_ssl` package. This package is shipped with RedHat Enterprise Linux. Also, to build `pwauth` you need the `pam-devel` package. Use the following command to install `mod_ssl` and the `pam-devel` packages:

```
# yum install mod_ssl pam-devel
```

You also need the `pwauth` package. It can be downloaded from the following location:

<http://code.google.com/p/pwauth>

2. The BGWS server runs under the `bgws` user account. For security reasons, the `pwauth` program is configured such that only `bgws` can run it. Create a `bgws` system account using the following command:

```
# useradd -c "Blue Gene Web Services" -r bgws
```

Add the `bgws` account to the Blue Gene/Q administrators group. The user must be an administrator so that the BGWS server can connect to the database and run service actions and diagnostics.

3. Un-tar the `pwauth` tar file previously downloaded in step 1:
 - a. Change into the source directory and edit the `config.h` file making the following updates:
 - i. Comment out `SHADOW_SUN`
 - ii. Uncomment `PAM`
 - iii. Set `SERVER_UIDS` to the `bgws` user ID
 - b. Edit the `Makefile` (it is in the same directory as `config.h`) making the following changes:
 - i. Comment out this line:


```
#LIB= -lcrypt
```
 - ii. Set the `LIB` parameter as follows:


```
LIB=-lpam -ldl
```
 - c. Build `pwauth` and install it to `/usr/local/libexec` with the correct permissions using the following commands:


```
$ make
# cp pwauth /usr/local/libexec/pwauth
# chown root:bgqadmin /usr/local/libexec/pwauth
# chmod 4750 /usr/local/libexec/pwauth
```
 - d. `pwauth` in PAM mode uses the “`pwauth`” PAM stack. You must create a `pwauth` pam config in `/etc/pam.d`:


```
# cd /etc/pam.d
# ln -s system-auth pwauth
```

Now the system is set up so that the BGWS server can run. The BGWS server is typically started by `BGmaster`. You might want to make further changes to the configuration, as described in the next section. For example, you might want to set the `machine_name` and allow connections from front end nodes by changing the `listen_ports`.

To support the Blue Gene Navigator web application, the Apache `httpd` server is typically configured to forward requests for the BGWS to the BGWS server. Setting up the `httpd` server is documented in the Chapter 2, “Navigator” on page 7.

15.2 Configuration

The BGWS server can be configured by setting options in the Blue Gene/Q configuration file (`/bgsys/local/etc/bg.properties`) or by setting command-line options when the server starts. The BGWS server is typically started under BGmaster. The executable is `/bgsys/drivers/ppcfloor/sbin/bgws_server`.

15.2.1 Configuration file options

Configuration options used by the BGWS server are in the `[bgws]` section of the Blue Gene/Q configuration file (`/bgsys/local/etc/bg.properties`). Configuration options are:

- ▶ `machine_name = My Blue Gene`
User interfaces, such as the Blue Gene Navigator, can display this option so that users know what machine they are connected to. Defaults to the system host name. This value can be refreshed by running the command `refresh_config bgws_server` from `bg_console`.
- ▶ `listen_ports = [::1]:32071,127.0.0.1:32071`
The ports that the BGWS server will listen for connections on. The format is a comma-separated list of host:port pairs. Host is an IP address or host name. IP addresses containing colons must be enclosed in brackets `[]`. Port is a port number or service name. The host and port are optional. If the host is not specified the server will listen on any interface. If the port is not specified the default port is used (32071).
- ▶ `path_base = /bg`
Path on the web server to the BGWS resources. Defaults to `/bg`. This value can be refreshed by running the command `refresh_config bgws_server` from `bg_console`.
- ▶ `thread_pool_size = auto`
Number of threads to run in the server. Allowed values are integers greater than 0 or *auto*. Specifying *auto* means the BGWS server picks the number of threads to start. The default is *auto*.
- ▶ `connection_pool_size = 20`
Maximum number of connections held in the BGWS server database connection pool.
- ▶ `extra_connection_parameters =`
Extra parameters to use when connecting to the database. Format is `NAME=VALUE` pairs separated by a semicolon (`;`). Optional, default is no extra parameters.
- ▶ `session_timeout = 3600`
Number of seconds of inactivity before a session is discarded. The default is 3600. This value can be refreshed by running the command `refresh_config bgws_server` from `bg_console`.
- ▶ `user_authentication_exe = /usr/local/libexec/pwauth`
Path to the program to run to authenticate users. The default is `/usr/local/libexec/pwauth`. This value can be refreshed by running the command `refresh_config bgws_server` from `bg_console`.
- ▶ `check_user_admin_exe = /bgsys/drivers/ppcfloor/bgws/libexec/checkUserAdmin`
Path to the program to run to check if a user is an administrator. If not set, all users will be non-administrators. This value can be refreshed by running the command `refresh_config bgws_server` from `bg_console`.

- ▶ `diagnostics_exe = /bgsys/drivers/ppcfloor/diags/bin/rundiags.py`
Path to the program to run for diagnostics. The default is `/bgsys/drivers/ppcfloor/diags/bin/rundiags.py`. This value can be refreshed by running the command **refresh_config bgws_server** from `bg_console`.
- ▶ `service_action_exe = /bgsys/drivers/ppcfloor/baremetal/bin/ServiceAction`
Path to the program to run for service actions. The default is `/bgsys/drivers/ppcfloor/baremetal/bin/ServiceAction`. This value can be refreshed by running the command **refresh_config bgws_server** from `bg_console`.

The BGWS server also gets configuration options from the `[security.ca]`, `[security.admin]`, and `[logging]` sections of the configuration file.

15.2.2 Command-line options

The BGWS server accepts the following options on the command line:

- listen-port arg** Ports to accept connections on. This overrides the `listen_ports` option in the `[bgws]` section of the configuration file. The format for this option is the same as the format for `listen_ports` in the `[bgws]` section of the `bg.properties` file.
- properties** This option provides the path to an alternate Blue Gene/Q configuration file. The default file, `/bgsys/local/etc/bg.properties` or the location specified by the `BG_PROPERTIES_FILE` environment variable if defined is used if the option is not specified.
- verbose** Sets the logging configuration. Setting these values overrides the logging configuration in the `bg.properties` file. The `--verbose` option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the `ibm.utility` logger to the `debug` value, the following command can be used:
bgws_server --verbose=DEBUG --verbose=ibm.utility=DEBUG
- h [--help]** Prints the help text

15.3 Utilities

Several utility programs are shipped with Blue Gene/Q that use the BGWS server. The following sections describe configuring the utilities and the utility programs.

15.3.1 Configuration

The BGWS server utilities and other BGWS commands read their configuration from the `[bgws.commands]` section in the Blue Gene/Q configuration file (`/bgsys/local/etc/bg.properties`). The following setting is configured in this file:

```
base_url = https://localhost:32071/bg
```

This is the base URL that the commands connect to. The default is `https://localhost:32071/bg`. The BGWS server must accept requests on that URL. If you want users to be able to run commands, such as `list_jobs`, from front end nodes you must change the host name to your service node.

In addition to the option in the [bgws.commands] section, the utilities and BGWS commands read configuration options from the [security] and [logging] sections in the configuration file.

15.3.2 The `bgws_server_status` command

The `bgws_server_status` command displays the status of the BGWS server.

Example 15-1 shows sample output when the BGWS server is running.

Example 15-1 Output of `bgws_server_status`

```
$ bgws_server_status
BGWS server is running and available.
```

Example 15-2 shows sample output when including details (-d).

Example 15-2 Output with details

```
$ bgws_server_status -d
BGWS server is running and available.
Server started: 2011-10-27 12:59:57.572908
Requests in progress: 1
Requests handled: 327
Max request time: 82.0508
Avg request time: 11.0661
```

The `bgws_server_status` command accepts the following command-line options:

- | | |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -d [--details] | Prints server status details. |
| --properties | Provides the path to an alternate Blue Gene/Q configuration file. The default file, <code>/bgsys/local/etc/bg.properties</code> or the location specified by <code>BG_PROPERTIES_FILE</code> environment variable if defined is used if the option is not specified. |
| --base | Overrides the <code>base_url</code> in the [bgws.commands] section of the <code>bg.properties</code> file. Refer to the documentation in section 15.3.1, “Configuration” on page 180. |
| --verbose | Sets the logging configuration. Setting these values overrides the logging configuration in the <code>bg.properties</code> file. The <code>--verbose</code> option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the <code>ibm.utility</code> logger to the <code>debug</code> value, the following command can be used:
<code>bgws_server_status --verbose=DEBUG --verbose=ibm.utility=DEBUG</code> |
| -h [--help] | Print help text. |

The user must be an administrator to use this command.

15.3.3 The `bgws_server_refresh_config` command

This utility tells the BGWS server to re-read the configuration file and continue running with the new values. Some of the values in the `bg.properties` file can be updated dynamically. The documentation for the value will say if the value can be updated by refreshing the configuration of the BGWS server with this command.

This command outputs nothing if the request was successful.

This command allows one optional positional parameter, the filename of the new `bg.properties` file, to read. If the filename is not present, the BGWS server will re-read its current configuration file. If the filename is an empty string, the BGWS server will re-read the default configuration file (`/bgsys/local/etc/bg.properties`). Otherwise, the provided filename will be read.

The `bgws_server_refresh_config` command also allows the following options:

- properties arg** This option provides the path to an alternate Blue Gene/Q configuration file. The default file, `/bgsys/local/etc/bg.properties` or the location specified by `BG_PROPERTIES_FILE` environment variable if defined is used if the option is not specified.
- base** This option overrides the `base_url` in the `[bgws.commands]` section of the `bg.properties` file. Refer to the documentation in section 15.3.1, “Configuration” on page 180.
- verbose** Sets the logging configuration. Setting these values overrides the logging configuration in the `bg.properties` file. The `--verbose` option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the `ibm.utility` logger to the `debug` value, the following command can be used:
bgws_server_refresh_config --verbose=DEBUG
--verbose=ibm.utility=DEBUG
- h [--help]** Print help text.

The user must be an administrator to use this command.

15.3.4 The `bgws_server_log_level` command

The `bgws_server_log_level` command changes the run time logging configuration of the BGWS server. The positional parameters to the `bgws_server_log_level` command are new settings for the different loggers.

The `bgws_server_log_level` command accepts the following command-line options:

- properties** Provides the path to an alternate Blue Gene/Q configuration file. The default file, `/bgsys/local/etc/bg.properties` or the location specified by `BG_PROPERTIES_FILE` environment variable if defined is used if the option is not specified.
- base** Overrides the `base_url` in the `[bgws.commands]` section of the `bg.properties` file. Refer to the documentation in section 15.3.1, “Configuration” on page 180.

- verbose** Sets the logging configuration. Setting these values overrides the logging configuration in the `bg.properties` file. The `--verbose` option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the `ibm.utility` logger to the *debug* value, the following command can be used:
- ```
bgws_server_log_level --verbose=DEBUG
--verbose=ibm.utility=DEBUG
```
- h [--help]** Prints the help text.

The user must be an administrator to use this command.





## Real-time server

The Blue Gene/Q Real-time server monitors the database transaction logs and sends events to any connected clients.

## 16.1 Real-time server

The Real-time server provides real-time notifications of system events to client applications. Applications use the Real-time API, provided as part of the Blue Gene Scheduler APIs, to receive these notifications. By using these notifications rather than polling the database on an interval, applications can be more efficient and responsive.

The types of system events that can be monitored using the Real-time APIs are block status changes (for example, booting, freeing), job status changes (for example, loading, starting, terminating), hardware state changes (becoming available or unavailable), and RAS events.

The Real-time server works by monitoring the database transaction logs using Linux's inotify API. When the transaction log file changes, the server reads the changes using APIs provided by IBM DB2®. The changes are sent to the clients that are connected.

The server executable is `/bgsys/drivers/ppcfloor/sbin/bg_realtime_server`. The server is usually started by BGmaster.

## 16.2 Configuration

The Real-time server can be configured by setting options in the Blue Gene/Q configuration file or by setting command-line options when the server starts.

### 16.2.1 Configuration file options

Configuration options used by the Real-time server are in the `[realtime.server]` section of the Blue Gene/Q configuration file (`/bgsys/local/etc/bg.properties`). Configurable options are:

- ▶ `listen_ports = localhost:32061`

The ports that the Real-time server will listen for connections on. The format is a comma-separated list of `host:port` pairs. *Host* is an IP address or host name. IP addresses containing colons must be enclosed in square brackets, `[]`. *Port* is a port number or service name. The host and port are optional. If the host is not specified, the server will listen on any interface. If the port is not specified, the default port is used (32061).

- ▶ `command_listen_ports = localhost:32062`

The ports that the Real-time server will listen for command connections on. The format is a comma-separated list of `host:port` pairs. *Host* is an IP address or host name. IP addresses containing colons must be enclosed in square brackets `[]`. *Port* is a port number or service name. The host and port are optional. If the host is not specified, the server will listen on any interface. If the port is not specified, the default port is used (32062).

- ▶ `workers = 0`

The *workers* parameter sets the number of worker threads to start. 0 indicates to let the server pick; otherwise, it must be a positive number.

- ▶ `maximum_transaction_size = 40`

The number of operations in a transaction before the server enters large-transaction mode. When the server is in large-transaction mode, clients will not receive real-time events. The default is 40.

The Real-time server also gets configuration options from the `[security.ca]`, `[security.admin]`, and `[logging]` sections of the configuration file.

## 16.2.2 Command-line options

The Real-time server accepts the following options on the command line:

### **--listen-port arg**

Ports to accept connections on. This overrides the `listen_ports` option in the `[realtime.server]` section of the configuration file. The format for this option is the same as the format for `listen_ports` option in the `[realtime.server]` section of the `bg.properties` file.

### **--command-listen-port arg**

Ports to accept command connections on. This overrides the `command_listen_ports` option in the `[realtime.server]` section of the configuration file. The format for this option is the same as the format for the `command_listen_ports` in the `[realtime.server]` section of the `bg.properties` file.

### **--properties arg**

This option provides the path to an alternate Blue Gene/Q configuration file. The default file, `/bgsys/local/etc/bg.properties` or the location specified by `BG_PROPERTIES_FILE` environment variable if defined is used if the option is not specified.

### **--verbose arg**

Sets the logging configuration. Setting these values overrides the logging configuration in the `bg.properties` file. The `--verbose` option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: `OFF`, `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG`, `TRACE`, or `ALL`. For example, if you want to set both the default logger and the `ibm.utility` logger to the `debug` value, the following command can be used:

```
bg_realtime_server --verbose=DEBUG --verbose=ibm.utility=DEBUG
```

### **-h [ --help ]**

Prints the help text.

## 16.3 Security

Each client must present the Blue Gene/Q administrative certificate when it connects to the Real-time server. The authentication process is handled automatically by the Real-time API and is transparent to the user. Clients not presenting the administrative certificate are rejected. Communication between the client and server is encrypted using secure sockets layer (SSL).

## 16.4 Utilities

Utilities are provided to get the status of the Real-time server and change the server's logging configuration.

## 16.4.1 Configuration

The Real-time server utilities read their configuration from the [realtime.server.command] section in Blue Gene/Q configuration file (/bgsys/local/etc/bg.properties). The following setting is configured in this file:

```
host = localhost:32062
```

The host and port to connect to. The Real-time server should be listening for command connections on the port. The format is a comma-separated list of host:port pairs. Host is an IP address or host name. IP addresses containing colons must be enclosed in square brackets []. Port is a port number or service name. The host and port are optional. If the host is not specified, the client will connect to localhost. If the port is not specified, the default port will be used (32062).

## 16.4.2 The realtime\_server\_status command

The `realtime_server_status` command displays the status of the Real-time server. The command executable is /bgsys/drivers/ppcfloor/sbin/realtime\_server\_status.

Example 16-1 shows sample output when the server has started and no clients connected.

*Example 16-1 Output of realtime\_server\_status with no clients*

---

```
Server status: running
Connected clients: 0
DB monitor state: idle
```

---

Example 16-2 shows the output of the command when one client has connected.

*Example 16-2 Output of realtime\_server\_status with one client*

---

```
Server status: running
Connected clients: 1
DB monitor state: monitoring
```

---

The `realtime_server_status` command accepts the following command-line options:

- host arg** Server to connect to. This option overrides the host setting in the [realtime.server.command] section of the configuration file. The format for this option is the same as the host setting in the [realtime.server] section of the bg.properties file.
- properties arg** This option provides the path to an alternate Blue Gene/Q configuration file. The default file, /bgsys/local/etc/bg.properties or the location specified by BG\_PROPERTIES\_FILE environment variable if defined is used if the option is not specified.
- verbose arg** Sets the logging configuration. Setting these values overrides the logging configuration in the bg.properties file. The --verbose option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the ibm.utility logger to the *debug* value, the following command can be used:  

```
realtime_server_status --verbose=DEBUG

--verbose=ibm.utility=DEBUG
```

`-h [ --help ]` Prints help text.

### 16.4.3 The `realtime_server_log_level` command

The `realtime_server_log_level` command changes the run-time logging configuration of the Real-time server. The positional parameters to the `realtime_server_logging` command are new settings for the different loggers.

The `realtime_server_log_level` command accepts the following arguments:

`--host arg` Server to connect to. This option overrides the host setting in the [realtime.server.command] section of the configuration file. The format for this option is the same as the host setting in the [realtime.server] section of the bg.properties file.

`--properties arg` This option provides the path to an alternate Blue Gene/Q configuration file. The default file, /bgsys/local/etc/bg.properties or the location specified by BG\_PROPERTIES\_FILE environment variable if defined is used if the option is not specified.

`--verbose arg` Sets the logging configuration. Setting these values overrides the logging configuration in the bg.properties file. The --verbose option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the ibm.utility logger to the *debug* value, the following command can be used:  
**`realtime_server_log_level --verbose=DEBUG`**  
**`--verbose=ibm.utility=DEBUG`**

`-h [ --help ]` Prints help text.

## 16.5 Requirements

Because of the way the Real-time server works, it has a couple of requirements that are unique to it beyond the other servers provided as part of the Blue Gene/Q system. It must be able to read the database transaction logs, and the database must be configured to support reading the transaction logs.

### 16.5.1 Access to the database transaction logs

The Real-time server must reside on the same host as the Blue Gene/Q DB2 database and have read access to the database transaction logs. BGmaster is typically configured to start the Real-time server as the database user (bgqsysdb) so that it has access. If the Real-time server does not have access to the database transaction logs, the Real-time server will shut down with the error message shown in Example 16-3 when a client connects.

*Example 16-3 Message generated when permission to transaction logs denied*

---

```
(INFO) [0x7f7473301820] ibm.realtime.server.db2.DbChangesMonitor: Start monitoring because a client
connected.
(INFO) [0x7f7473301820] ibm.realtime.server.db2.DbChangesMonitor: db2CfgGet sql_rc=0 sqlcode=0
(INFO) [0x7f7473301820] ibm.realtime.server.db2.DbChangesMonitor: log path for 'BGDB0' is
'/dbhome/bgqsysdb/bgqsysdb/NODE0000/SQL00003/SQLQLOGDIR/'
(WARN) [0x7f7473301820] ibm.realtime.server.db2.DbChangesMonitor: Client rejected because an error
occured. The error is Permission denied
```

./bg\_realtime\_server: failed, Permission denied

---

## 16.5.2 Database configuration

The database must be configured with archive logging. The DB2 APIs used by the Real-time server require that the database is configured with archive logging. The database will use archive logging if the LOGARCHMETH1 configuration option is not OFF. If the database is not configured with archive logging, the Real-time server will shut down when a client connects.

The database tables that the Real-time server monitors must have DATA CAPTURE CHANGES enabled. By default, at install time, this is done by the script that creates the database schema, /bgsys/drivers/ppcfloor/db/schema/createBGQSchema.

Use the following steps to verify that the current settings are correct:

1. Change the current user to bgqsysdb and supply the password when prompted.

```
$ su - bgqsysdb
```

2. Run the following command to view the current database configuration:

```
$ db2 "get database configuration for bgdb0"
```

3. If the database is set to the default install configuration, you will see the following line in the database configuration output:

```
First log archive method (LOGARCHMETH1) = DISK:/dbhome/bgqsysdb/dblogs/archive/bgdb0
```

If the output shows LOGARCHMETH1 = OFF, the database must be reconfigured. To configure the database with archive logging, after creating the database and before running createBGQSchema, perform the following steps:

1. Switch the user to bgqsysdb, and update the database configuration with the following commands:

```
su - bgqsysdb
```

```
mkdir -p /dbhome/bgqsysdb/dblogs/archive/bgdb0
```

```
db2 "update database configuration for bgdb0 using logarchmeth1
```

```
DISK:/dbhome/bgqsysdb/dblogs/archive/bgdb0"
```

**The update command:** The DB2 `update` command in step 1 extends over two lines; however, it is a single command

2. Stop and restart DB2 so the changes will take effect:

```
db2 force application all
```

```
db2stop
```

```
db2start
```

3. After the database restarts, you must make a backup. Use the command:

```
db2 "BACKUP DATABASE BGDB0"
```

4. Create a cron job (logged in as root) to remove old archive log files. Create a file /etc/cron.daily/bluegene-db-log-cleaner.sh with the contents shown in Example 16-4 on page 191.



*Example 16-4 Contents of bluegene-db-log-cleaner.sh*

---

```
#!/bin/bash
find /dbhome/bgqsysdb/dblogs/archive/bgdb0/ -type f -mtime +1 -exec rm {} \;
```

---

5. Make sure the new file is executable by running the following command:

```
chmod +x /etc/cron.daily/bluegene-db-log-cleaner.sh
```





## Distributed Control System

This chapter describes a centralized load balancing cluster for distributing service node bandwidth and CPU utilization over multiple smaller nodes. The centralized model focuses on splitting the `mc_server` process and managing it with a distributed process manager.

Two types of clustering models are supported on the Blue Gene/Q system: service node failover and load balancing. Service node failover provides redundancy to minimize resource downtime. Load balancing clusters distribute processing to multiple server nodes to improve resource utilization. The two solutions (service node failover and load balancing) can be used separately or together.

The Blue Gene/Q Service Node Failover using Linux High Availability solution is designed to provide transparent server failover for the primary service node. This feature is installed as a customer option. It is covered in detail in *IBM System Blue Gene Solution: Blue Gene/Q Service Node Failover using Linux High Availability*, REDP-4657 IBM Redpapers™ publication.

## 17.1 Overview

Performance measurements show that the `mc_server` component of the Control System is the largest consumer of CPU and networking bandwidth, especially during block booting. Offloading the management of the Blue Gene/Q service network and hardware significantly reduces the load on the primary service node. This offload can be accomplished by distributing the management of the network connections on a single service node or adding subnet service nodes.

## 17.2 Distributed Control System software

The Control System allows the boot image distribution and other `mc_server` functions to be distributed over multiple systems. The software components that comprise the distributed machine controller are:

|                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>mc_server</b>  | The <code>mc_server</code> 's controlling process handles client connections and hardware arbitration. It coordinates the worker processes and dispatches work to them. To distribute this function, the machine controller is broken up further into the machine controller (MC) library and independent remote processes called <code>SubnetMc</code> . The <code>mc_server</code> is described in detail in Chapter 13, "Machine controller server" on page 159 |
| <b>MC library</b> | The <code>mc_server</code> is a server process built on the machine controller library. The machine controller library provides the low-level hardware control and communications.                                                                                                                                                                                                                                                                                 |
| <b>SubnetMc</b>   | The worker processes (instances of <code>SubnetMc</code> ) are responsible for managing the network connections and the Blue Gene/Q hardware assigned to them.                                                                                                                                                                                                                                                                                                     |
| <b>BGmaster</b>   | The <code>BGmaster</code> is the process manager on Blue Gene/Q for the Control System servers. Its purpose is to act as a centralized failure and recovery process. <code>BGmaster</code> is responsible for managing the <code>SubnetMc</code> components. <code>BGmaster</code> is described in detail in Chapter 11, "BGmaster" on page 129.                                                                                                                   |

## 17.3 Hardware configurations

A small configuration, such as the one shown in Figure 17-1, typically runs on a single service node with only one instance of the SubnetMc process running. This single process provides communication between the mc\_server process and the hardware. The amount of overhead generated by a single four-rack row of compute racks does not cause any network or CPU issues on the service node.

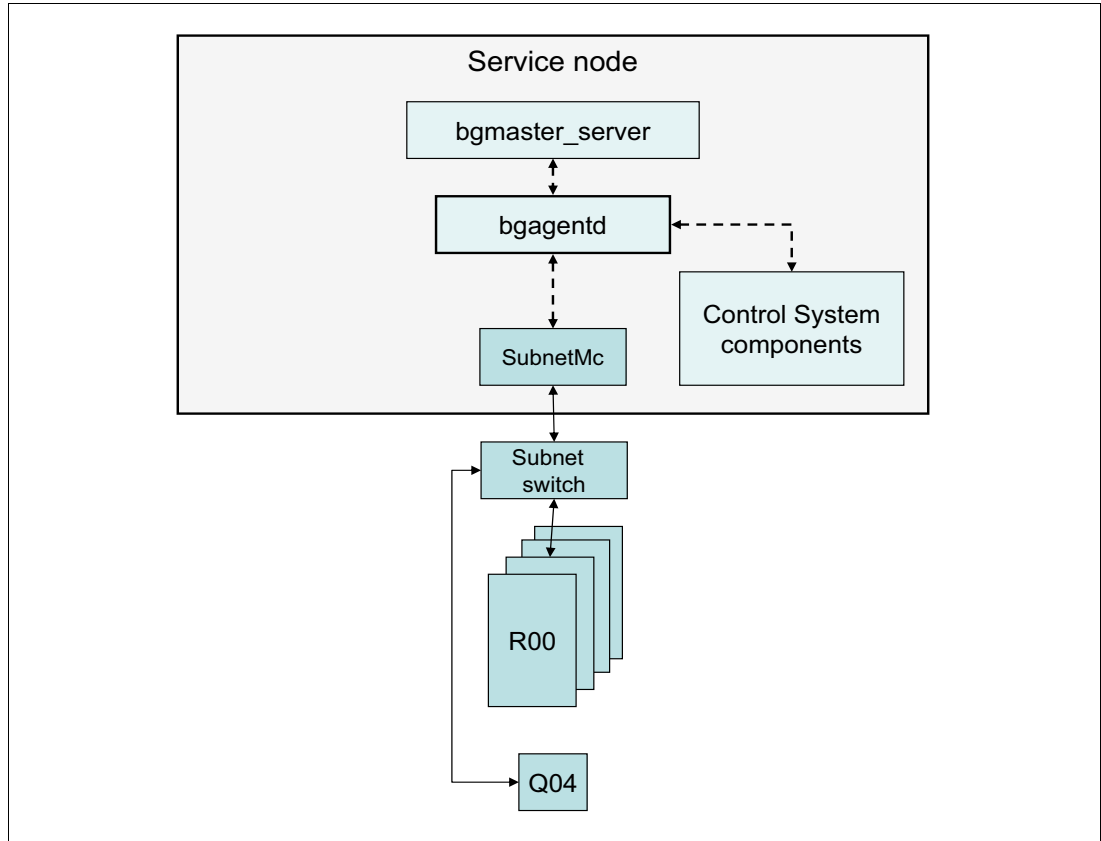


Figure 17-1 Small service node Distributed Control System system configuration

However, as additional compute and I/O racks are added, the load on the service node's CPU increases and network traffic can cause delays in communication with the hardware. To deal with the additional stress, configuration changes to software or hardware can be made to relieve the burden on the service node. In addition to a single service node with one SubnetMc process running, several other configurations are supported. These configurations include a service node hosting multiple SubnetMc processes, a service node with multiple subnet service nodes each running their own SubnetMc process, and a service node with multiple subnet service nodes each running their own SubnetMc process with ability to failover to backup subnet service nodes.

### 17.3.1 Multiple SubnetMc processes on a service node

Figure 17-2 on page 196 shows a service node running multiple SubnetMc processes to balance the load between the rows of compute and I/O racks. In this configuration, the SubnetMc processes exist on the service node and are managed by a single bgagentd process.

In Figure 17-2, there are twelve Blue Gene/Q racks set up in three rows of four racks. Each row is configured as a separate subnet that consists of four Blue Gene racks, an I/O rack (Qxx), and the subnet switch.

The load is balanced by spreading the Ethernet traffic over multiple interfaces and splitting the SubnetMc work among three processes.

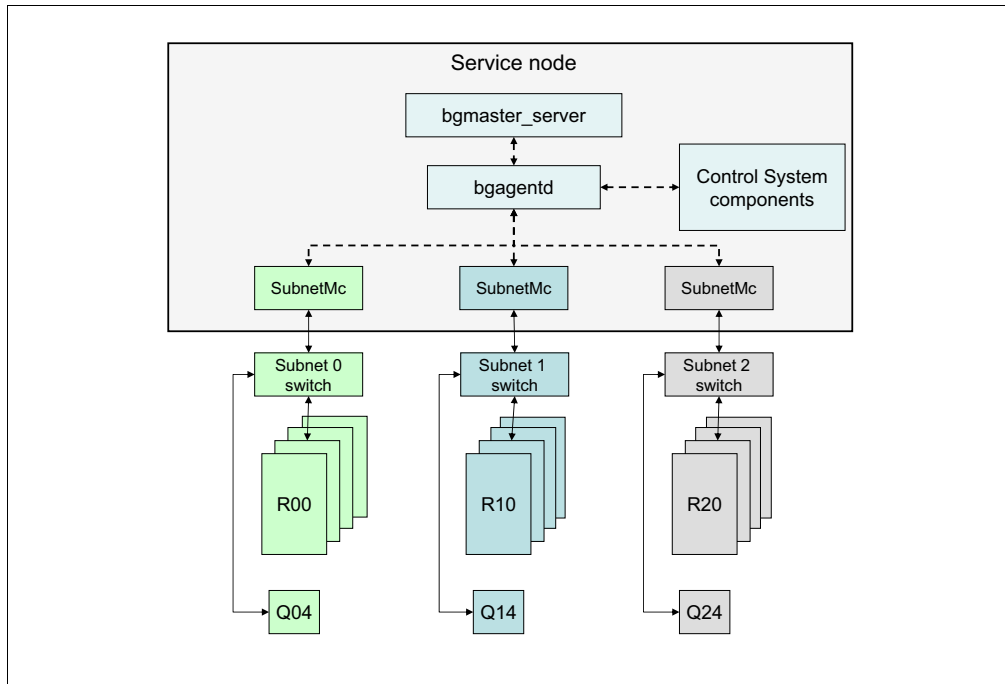


Figure 17-2 Multiple SubnetMc processes on a service node

### 17.3.2 Multiple subnet service nodes

The hardware solution to reducing the load on the service node is shown in Figure 17-3 on page 197. In this configuration, the mc\_server process runs on the primary service node. The additional rows of compute and I/O racks are grouped by row into subnets. Each of the subnets are controlled by their own subnet service node. The subnet service node also has a bgagentd process and a SubnetMc process running. In this configuration the bgagentd on the subnet service nodes communicates with bgmaster\_server on the primary service node.

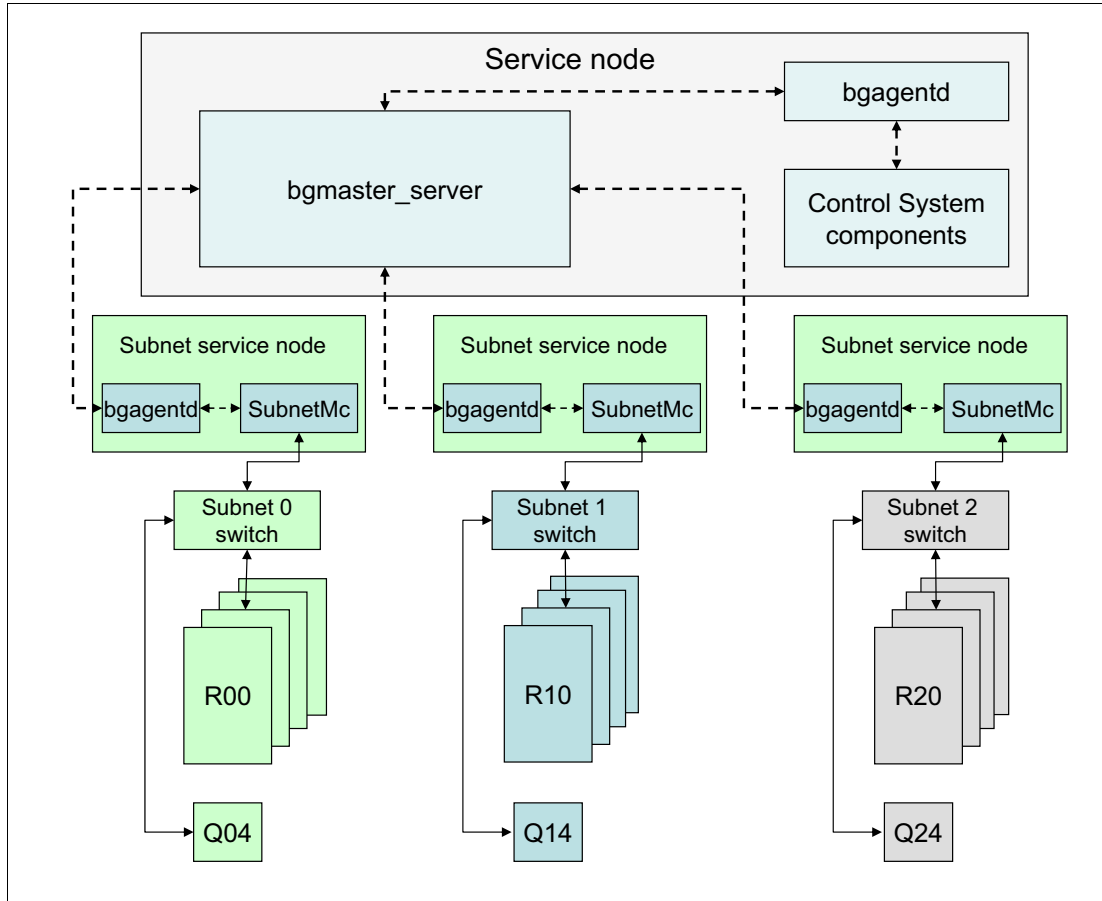


Figure 17-3 Distributed configuration with multiple subnet service nodes

### 17.3.3 Multiple subnet service nodes with failover

This hardware solution is similar to the setup in Figure 17-3 on page 197 but with failover enabled. When a subnet service node failure is detected, the bgmaster\_server process running on the service will failover the SubnetMc process to a backup subnet service node. As part of the failover processing, the SubnetMc processes connections with the Blue Gene/Q hardware. The hardware in this scenario remains booted and usable.

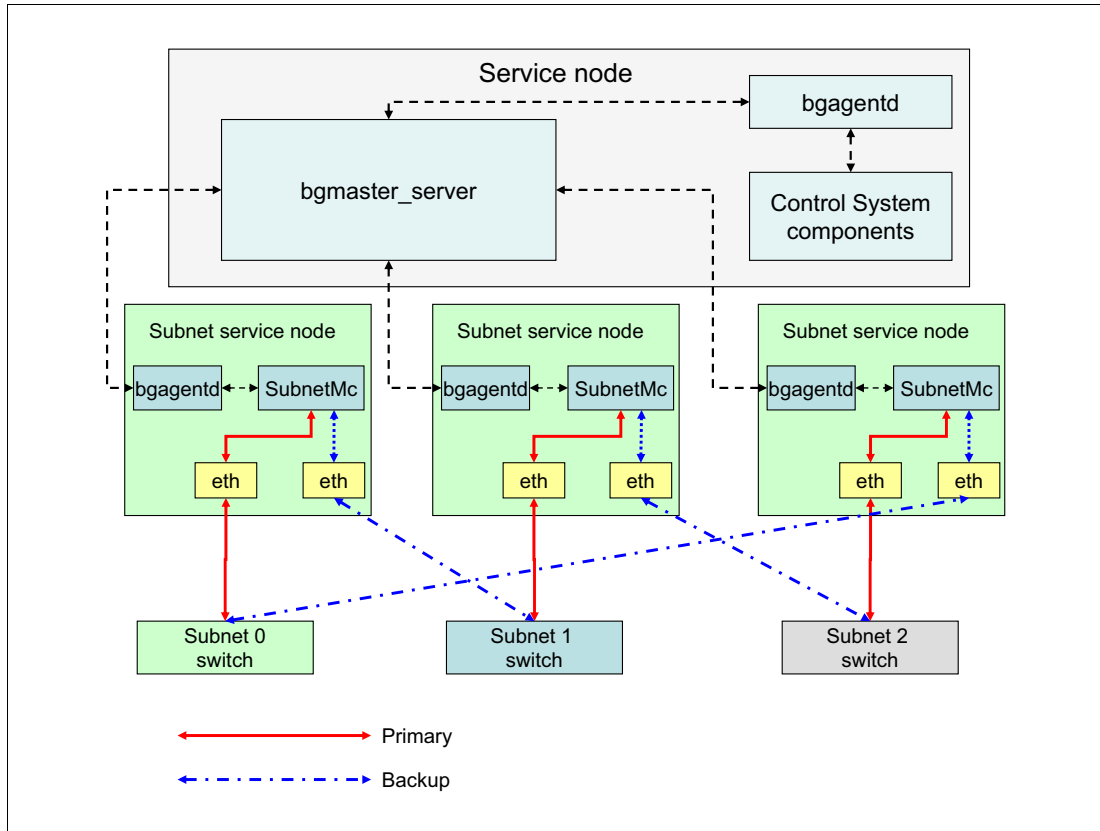


Figure 17-4 Multiple subnet service nodes with failover



## 17.4 SubnetMc configuration

Whether the environment has a single subnet or multiple subnets a SubnetMc must be configured. The SubnetMc configuration is done in the [machinecontroller.subnet.x] section of the /bgsys/local/etc/bg.properties file (where x is the number of the subnet). Configurable options for SubnetMc are (default values are shown):

▶ [machinecontroller.subnet.0]

This type of header indicates the start of the parameters for the SubnetMc. In this case, it is the first SubnetMc in the configuration file. Subsequent subnets increment the number at the end of the header (for example, [machinecontroller.subnet.1]).

▶ Name = Subnet1

*Name* is the identifier (alias) for the SubnetMc. It represents a unique identifier for each subnet and is used when logging messages, RAS events, and so on. The alias name for each subnet MC process in the [master.binmap] section must match the *Name* keyword in the [machinecontroller.subnet.X] section. By default the first subnet name is "Subnet1", the second subnet name is "Subnet2", and so on. A unique alias name is needed for each instance of a SubnetMc process because the aliases must be associated with different agents and this name determines the subnet service node (host system) where each SubnetMc runs.

▶ PrimaryServer = 127.0.0.1

The *PrimaryServer* is the IP address of the machine (host) that runs this SubnetMc process. If there is only one subnet, the *PrimaryServer* is set to the 127.0.0.1 (loopback) address. In environments with multiple subnet servers, this address is the interface that the service node uses to contact the subnet service node. Typically this address is a 192.169.x.x address.

▶ BackupServer = 127.0.0.1

The *BackupServer* is the IP address of the machine (host) that runs this SubnetMc process when the primary server is unavailable. If there is only one subnet, the *BackupServer* is set to the 127.0.0.1 (loopback) address. In environments with multiple subnet servers, this address is the interface that the service node uses to contact the subnet service node. Typically this address is a 192.169.x.x address.

▶ PrimaryServerInterface = lo

The *PrimaryServerInterface* is the Ethernet adapter that the mc\_server uses to communicate with the SubnetMc for this subnet. In a single service node configuration, this parameter is set to *lo* (local loopback). When multiple subnet service nodes are being used, the *PrimaryServerInterface* designates the address that is used to connect the service node to the subnet service node. Typically it is an adapter with a 192.169.x.x address.

▶ BackupServerInterface = lo

The *BackupServerInterface* is the Ethernet adapter that the mc\_server uses to communicate with the SubnetMc for this subnet when the primary server is unavailable. In a single service node configuration, this parameter is set to *lo* (local loopback). When multiple subnet service nodes are being used, the *BackupServerInterface* designates the address that is used to connect the service node to the subnet service node. Typically it is an adapter with a 192.169.x.x address.

▶ PrimaryServerPort = 33456

The *PrimaryServerPort* is the TCP port number that the SubnetMc for this subnet is listening on. The default is 33456.

- ▶ BackupServerPort = 33456

The *BackupServerPort* is the TCP port number that the SubnetMc for this subnet is listening on when the primary server is unavailable and the backup server is used instead. The default is 33456.

- ▶ PrimaryBGInterface = eth1

The *PrimaryBGInterface* defines the Ethernet adapter on the subnet service node used to talk to the Blue Gene hardware on the service network. The interface IP address is in the 10.x.x.x range, and the second octet increments with the number of rows in the system. A single SubnetMc system has addresses 10.0.x.x. A second SubnetMc process has the address 10.1.x.x.

- ▶ BackupBGInterface = eth1

The *BackupBGInterface* defines the Ethernet adapter on the subnet service node used to talk to the Blue Gene hardware on the service network when the primary server is unavailable and the backup server is used instead. The interface IP address is in the 10.x.x.x range, and the second octet increments with the number of rows in the system. A single SubnetMc system has the address range 10.0.x.x. A second SubnetMc has the address range 10.1.x.x.

- ▶ HardwareToManage = R00

*HardwareToManage* is a list of the racks that are allowed to communicate on this subnet (for example, the racks expected to be on this subnet). This list includes both the compute racks and the I/O racks.

There are several other sections of the `bg.properties` file that might need to be updated depending on the number of subnets defined.

The `[master.binmap]` section defines the binary used for the SubnetMc. Additional subnets use the same binary. Example 17-1 shows a system that has two SubnetMcs defined.

*Example 17-1 [master.binmap] section*

---

```
[master.binmap]
Maps an alias to a binary path
...
Subnet1=/bgsys/drivers/ppcfloor/control/sbin/SubnetMc_64
Subnet2=/bgsys/drivers/ppcfloor/control/sbin/SubnetMc_64
...
```

---

The `[master.binargs]` section defines the command-line arguments. In this case, as shown in Example 17-2, different command-line arguments are needed for the two SubnetMc processes.

*Example 17-2 [master.binargs] section*

---

```
[master.binargs]
List of arguments for each alias specified in [master.binmap] section
...
Subnet1=--ip 172.16.1.135,172.16.1.136 --inport 33456
Subnet2=--ip 172.16.1.136,172.16.1.135 --inport 33457
...
```

---

Assign the SubnetMc aliases to hosts in the `[master.policy.host_list]` section. It is important to correctly match the alias with the host IP addresses where the binary can run. In this case, as

shown in Example 17-3, each SubnetMc has a preferred host where it can run and a backup (failover) host where it can run.

*Example 17-3 [master.policy.host\_list] section*

---

```
[master.policy.host_list]
Comma separated list of hosts on which the binary associated with the alias may start
....
Subnet1=172.16.1.135,172.16.1.136
Subnet2=172.16.1.136,172.16.1.135
...
```

---

In the [master.policy.failure] section, define the failure policy. For more information about policies, see Chapter 11, “BGmaster” on page 129. In this case, as shown in Example 17-4, a failover policy is defined for each SubnetMc process.

*Example 17-4 [master.policy.failure] section*

---

```
[master.policy.failure]
Format:
policy_name=trigger,[failover|restart|cleanup],<retries>,<failover_from:failover_to>|etc
...
Subnet1_failover=agent,failover,3,172.16.1.135:172.16.1.136|172.16.1.136:172.16.1.135
Subnet2_failover=agent,failover,3,172.16.1.136:172.16.1.135|172.16.1.135:172.16.1.136
...
```

---

The [master.policy.map] section, Example 17-5, sets the policy for the SubnetMc processes.

*Example 17-5 [master.policy.map] section*

---

```
[master.policy.map]
Map comma separated list of named policies to an alias
...
Subnet1=Subnet1_failover
Subnet2=Subnet2_failover
...
```

---

## 17.4.1 Troubleshooting configuration issues

The following items are some possible configuration problems that can lead to confusing or inconsistent behavior:

- ▶ Under certain conditions, using *localhost* instead of 127.0.0.1 can result in a SubnetMc process failing to start or restart. If the *PrimaryServer* or *BackupServer* keywords are set to *localhost* in the [machinecontroller.subnet.x] sections, change them to 127.0.0.1.
- ▶ If a SubnetMc has no backup service network connections available, change the *BackupServer*, *BackupServerInterface*, *BackupServerPort*, and *BackupBGInterface* settings to match the *PrimaryServer*, *PrimaryServerInterface*, *PrimaryServerPort*, and *PrimaryBGInterface* settings. Repeat this step for all configured subnets.
- ▶ Cross-check the [master.policy.host\_list] section to confirm that the alias configured for this subnet can only start on the IP addresses listed from the *PrimaryServer* and *BackupServer*. Repeat this step for all configured subnets.





# Security

This chapter provides an overview of the security architecture for Blue Gene/Q. There are three aspects to the Blue Gene/Q security model: an object-based model, network communications, and the database.

## 18.1 Object based model

Blue Gene/Q uses an object-based security model with three basic object types:

- ▶ Job
- ▶ Block
- ▶ Hardware

There are five types of actions upon these objects:

- ▶ Create
- ▶ Read
- ▶ Update
- ▶ Delete
- ▶ Execute

Not all actions apply to all object types, for example, the update action does not apply for a job object.

There is also a special value of *All* that when configured in the `bg.properties` configuration file allows a user or group the authority to perform any of the above actions.

User and group authorities for objects can be defined in the `bg.properties` file or granted and revoked through `bg_console`. The Blue Gene/Q Scheduler API also provides interfaces for job schedulers to control user access to compute blocks so they can run jobs. The job and block authority commands that are available from `bg_console` are: **grant\_block\_authority**, **revoke\_block\_authority**, **list\_block\_authority**, **grant\_job\_authority**, **revoke\_job\_authority**, and **list\_job\_authority**. For more information about these commands, see Chapter 5, “Control System console” on page 67.

Security configuration in the `bg.properties` file is kept in the `[security.jobs]`, `[security.hardware]`, and `[security.blocks]` sections. Both user and group IDs are supported in defining the permissions. Example 18-1 shows an example of the `[security.jobs]` and `[security.blocks]` sections. In this example, user `bgqadmin` or any user in the `bgqadmin` group is allowed to perform any action on any job, on any hardware or any block.

### *Example 18-1 bg.properties object security*

---

```
[security.blocks]
All of the values in this section can be refreshed by running refresh_config from
bg_console.

all = bgqadmin
Comma separated list of users or groups allowed to execute all actions on blocks.

create =
Comma separated list of users or groups allowed to create blocks.

read =
Comma separated list of users or groups allowed to read blocks.

update =
Comma separated list of users or groups allowed to update blocks.

delete =
Comma separated list of users or groups allowed to delete blocks.

execute =
Comma separated list of users or groups allowed to execute blocks.
```

```
[security.jobs]
All of the values in this section can be refreshed by running refresh_config from
bg_console.

all = bgqadmin
Comma separated list of users or groups allowed to execute all actions on all jobs.

read =
Comma separated list of users or groups allowed to read all jobs.

execute =
Comma separated list of users or groups allowed to execute all jobs.

[security.hardware]
All of the values in this section can be refreshed by running refresh_config from
bg_console.

all = bgqadmin
Comma separated list of users or groups allowed to execute all actions on hardware.

read =
Comma separated list of users or groups allowed to read hardware.

execute =
Comma separated list of users or groups allowed to execute hardware.
```

---

## 18.2 Network communication security

Communication between the TCP/IP clients and servers provided by Blue Gene/Q are secured using the Secure Sockets Layer (SSL) protocol, as provided by the OpenSSL and Java libraries. This section describes how security is established.

### 18.2.1 Certificates and keys

Trust must be established between a client and server. Both sides must prove their identity, and both sides can only trust the data they receive if it cannot be modified along the way. The use of the SSL protocol and encryption ensures that the data is not modified.

Identity is established using certificates. If a server or client can present a certificate, it can prove that it has that identity. There are three identities that are used in the Blue Gene security model: certificate authority (CA), administrative, and command. A certificate is stored in two files: a public certificate file and a private key file.

Figure 18-1 on page 206 provides an overview of how security certificates are used to protect communication between clients and servers. The command and administrative certificates are signed by the CA. The servers use the administrative key to communicate between themselves and to communicate with commands, such as **bg\_console**. End-user commands use the command certificate. Administrative commands, such as diagnostics use the administrative certificate.

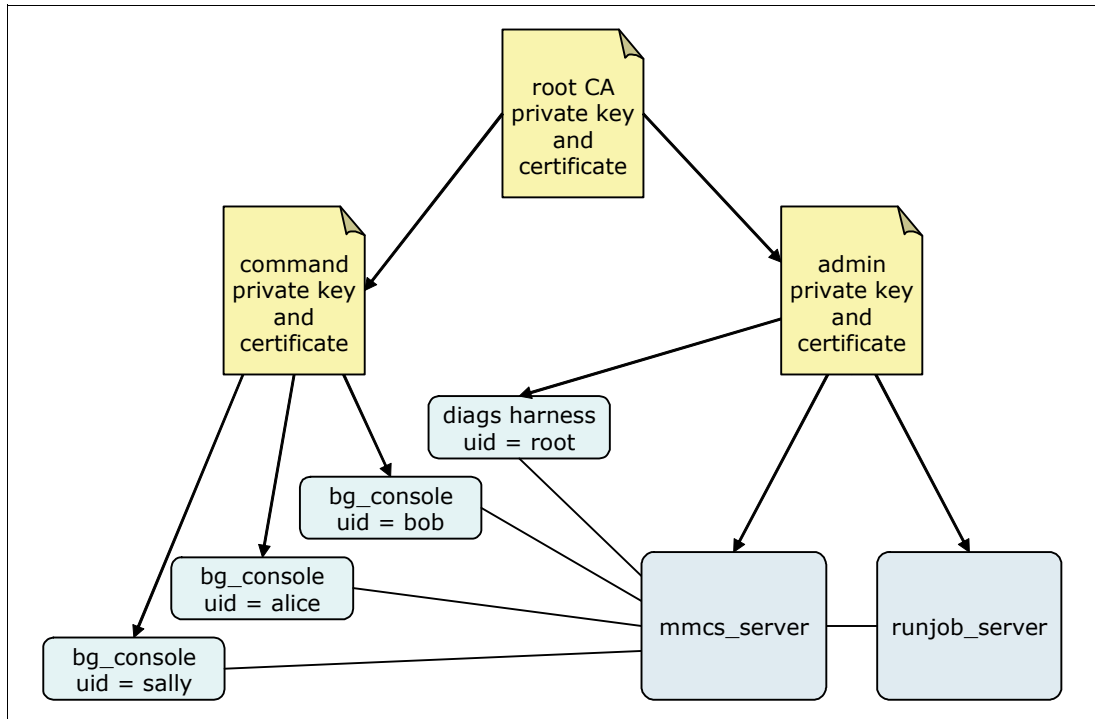


Figure 18-1 Certificates for trusted communication

The Certificate Authority (CA) certificate is used to sign the administrative and command certificates. If a certificate is signed with the CA certificate, it is considered valid; otherwise, it cannot be trusted. The Control System can trust that the CA verified the Common Name (CN) in the certificate. Every user must be able to read the CA public key file so that they can verify the certificates that they receive. Only a trustworthy user (root) should be able to access the CA private key file because this can be used to generate a new certificate.

The administrative certificate identifies Blue Gene/Q administrative users. If a user can read the administrative private key file, they are considered to be a Blue Gene/Q administrator.

The command certificate identifies the Blue Gene/Q end user command programs. The private key for this certificate should only be readable by the `bgqcommand` system account. The Blue Gene/Q end user command programs are `setuid bgqcommand` so that they can read this key. Only user `root` can make a program `setuid bgqcommand`. A normal user cannot subvert this security by running their own copy of the command.

When an end user command connects to the server, it sends the real user information (uid and groups). If a user can send a different user's user information, it can spoof the other user. Requiring access to the command private key allows the server to trust that the client is sending valid user information. Also, as a `setuid` program, the user cannot run it under `ptrace` and change the application in progress because they can lose their `bgqcommand` authority and cannot read the private key file.

## 18.2.2 Handshake

After a client connects to a server a handshake takes place. The server and client exchange certificates, and the client sends the user information before normal communication occurs. This section describes the handshake from both the client's and server's perspectives.



The typical client handshake follows these steps:

1. If the user has access to the administrative key file, it presents the administrative certificate; otherwise, it presents the command certificate.
2. Verifies that the certificate presented by the server is signed by the CA and that the certificate's CN matches the configured administrative CN.
3. The user's information is sent to the server.
4. Now the regular protocol is followed.

The typical server handshake follows these steps:

1. Presents the administrative certificate.
2. Verifies that the certificate presented by the client is signed by the CA.
3. If the CN in the client's certificate matches the configured administrative CN, the client is accepted as an administrator.
4. If the CN in the client's certificate matches the configured command CN, the user information is read.
5. If the certificate has neither the administrative CN nor the command CN, the connection is closed.
6. Now the regular protocol is followed.

There are variations in the handshake procedure. Some clients are administrative only, so it only attempts to use the administrative certificate (for example, the **refresh\_config** commands and diagnostics). Some servers only accept administrative connections, so it only accepts the administrative certificate (for example, the Real-time server).

### 18.2.3 Configuration

Certificate configuration in the `bg.properties` file is kept in the `[security.ca]`, `[security.admin]`, and `[security.command]` sections. Example 18-2 shows an example of the `[security.ca]`, `[security.admin]`, and `[security.command]` sections.

*Example 18-2 bg.properties certificate security*

---

```
[security.ca]
 # Certificate Authority.

certificate = /bgsys/local/etc/security/ca/cert.pem
 # File containing the root CA (certificate authority) certificate.
 # This property is optional, but either certificate or
 # certificates_dir must be set or use_default_paths must be 'true'.

certificates_dir = /bgsys/local/etc/security/root/
 # Directory containing root CA certificates.
 # This property is optional, but either certificate or
 # certificates_dir must be set or use_default_paths must be 'true'.

use_default_paths = false
 # Set to 'true' to use the system default root CA paths.
 # If this is 'false' then either certificate or certificates_dir
 # must be set.
```

```

keystore = /bgsys/local/etc/security/ca/cert.jks
 # File containing the root CA certificate in Java Keystore format.

[security.admin]
 # Administrative certificate.
certificate = /bgsys/local/etc/security/admin/cert.pem
 # File containing the Blue Gene administrative certificate.

key_file = /bgsys/local/etc/security/admin/key.pem
 # File containing the Blue Gene administrative private key.

cn = Blue Gene administrative
 # The certificate provided by the server or administrator must have this CN
 # (common name).

keystore = /bgsys/local/etc/security/admin/key.jks
 # File containing the Blue Gene administrative certificate in Java Keystore format
(jks).

[security.command]
 # Command certificate.

certificate = /bgsys/local/etc/security/command/cert.pem
 # File containing the Blue Gene command certificate.

key_file = /bgsys/local/etc/security/command/key.pem
 # File containing the Blue Gene command private key.

cn = Blue Gene command
 # The certificate provided by the Blue Gene command must have this CN.

```

---

## 18.2.4 Certificate file permissions

To establish effective trust between components, the private keys must be kept secret to prevent unauthorized users from escalating their privileges. The administrative key file is only readable by the administrative group. The command key file is readable by the bgqcommand system account.

Example 18-3 shows the correct file permissions for the various security keys.

### *Example 18-3 Permissions for keys*

---

```

bgqadmin@bgqsn etc> ll security/*
security/admin:
total 12
-rw-r--r-- 1 root root 859 Oct 26 17:39 cert.pem
-rw-r----- 1 root bgqadmin 1511 Oct 26 17:39 key.jks
-rw-r----- 1 root bgqadmin 887 Oct 26 17:39 key.pem
security/ca:

```

```

total 8
-rw-r--r-- 1 root root 636 Oct 26 17:39 cert.jks
-rw-r--r-- 1 root root 839 Oct 26 17:39 cert.pem
security/command:
total 8
-rw-r--r-- 1 root root 851 Oct 26 17:39 cert.pem
-rw----- 1 bgqcommand root 887 Oct 26 17:39 key.pem
bgqadmin@bgqsn etc>

```

---

## 18.2.5 Generating security keys and certificates

If your site already has a public key infrastructure in place, that can be used to create the administrative and command certificate files. The CN in the administrative certificate must match the configured administrative CN in the `bg.properties` file and the CN in the command certificate must match the configured command CN.

Alternatively, the Blue Gene/Q software ships with a simple utility that uses the OpenSSL command-line utility to generate certificates for use. The program is `/bgsys/drivers/ppcfloor/utility/sbin/generate_security_certificates.sh` and should be run from the `/bgsys/local/etc/security` directory by user `root`. Example 18-4 provides an example of how to use this program to generate security keys and certificates.

### *Example 18-4 Generating keys and certificates*

---

```

root@bgqsn etc>/bgsys/drivers/ppcfloor/utility/sbin/generate_security_certificates.sh
Generating RSA private key, 1024 bit long modulus
.....+++++
.....+++++
e is 65537 (0x10001)
Signature ok
subject=/C=US/ST=Minnesota/L=Rochester/O=IBM/OU=STG/CN=Root CA
Getting Private key
Certificate was added to keystore
Generating RSA private key, 1024 bit long modulus
.....+++++
.....+++++
e is 65537 (0x10001)
Signature ok
subject=/C=US/ST=Minnesota/L=Rochester/O=IBM/OU=STG/CN=Blue Gene administrative
Getting CA Private Key
security/admin/cert.pem: OK
Entry for alias 10241040677550452466cn=blue gene administrative, ou=stg, o=ibm, l=rochester,
st=minnesota, c=uscn=root ca, ou=stg, o=ibm, l=rochester, st=minnesota, c=us successfully
imported.
Import command completed: 1 entries successfully imported, 0 entries failed or cancelled
Generating RSA private key, 1024 bit long modulus
.....+++++
.....+++++
e is 65537 (0x10001)
Signature ok
subject=/C=US/ST=Minnesota/L=Rochester/O=IBM/OU=STG/CN=Blue Gene command
Getting CA Private Key

```

```
security/command/cert.pem: OK
root@bgqsn etc>
```

---

When this program runs, it deletes the certificate authority private key file. If the certificates are compromised in some way, run the program again to generate new certificates.

## 18.3 Database security

Multiple programs access the Blue Gene/Q DB2 database. The Blue Gene/Q servers, system configuration programs, and users of the Scheduler APIs are the primary users of the database. Access to the database is secured using standard DB2 security mechanisms. Blue Gene/Q administrative users are expected to have DBADM authority to the database.

Users with DBADM database authority have all privileges to the database objects and their contents. They can create, delete, and alter tables, update table contents, perform queries, and so on. DBADM authority can be assigned to individual users or groups. The following commands illustrate different ways in which you can give DBADM authority.

A DB2 user with SYSADM authority can grant other users access to the Blue Gene/Q database. One user with SYSADM authority is the database instance user, typically bgqsysdb.

Example 18-5 shows the DB2 command to grant DBADM authority to everyone in the bgqadmin group. The issuing user must be connected to the database before granting authority.

*Example 18-5 Granting group authority to DB2 database*

---

```
$ su - bgqsysdb
$ db2 connect to bgdb0
$ db2 grant dbadm on database to group bgqadmin
```

---

Example 18-6 shows the DB2 command to grant DBADM authority to user bgqadmin1. The issuing user must be connected to the database before granting authority.

*Example 18-6 Granting user authority to DB2 database*

---

```
$ db2 grant dbadm on database to user bgqadmin1
```

---



# Database

This chapter provides information about database configuration, maintenance, and optimization.

## 19.1 Database overview

All of the information about the Blue Gene/Q system's state is stored in the DB2 database. DB2 is installed and configured during the initial installation of the system. After it is installed, there is rarely a need for manual interaction with the database.

The data stored in DB2 is managed by the Blue Gene/Q Control System software. In all cases, the viewing and modification of the database information is handled through commands and interfaces, and never through direct database commands or SQL statements. For instance, all hardware status and operational state of the machine is represented in the database and can be viewed using the Blue Gene Navigator. If old data needs to be purged from any tables, this is accomplished using the dbPurge tool. See 19.3.1, "Purging old data from tables" on page 213. View the database as being the repository for machine information, but not requiring direct manipulation by users or administrators, and also not requiring deep knowledge of the DB2 product or SQL syntax.

## 19.2 Database configuration

The home directory for the database is /dbhome. Information about the local database configuration is maintained in the /bgsys/local/etc/bg.properties file. The default entries in the bg.properties file for the database are shown in Example 19-1.

*Example 19-1 bg.properties database entries*

---

```
[database]

name=BGDBO
 # The name of the database containing the Blue Gene schema.

#user=bgqsysdb
 # The user to connect to the database as.
 # This value is optional. If not set, then the current user is used.

#password=xxxxxx

schema_name=bgqsysdb
 # The database schema containing the Blue Gene tables.
 # This value is optional. If not set, the default schema is used.

computeRackRows=1
computeRackColumns=1
 # Used by dbPopulate to populate the schema
```

---

## 19.3 Database maintenance

There are two suggested steps to maintaining the database: purging old data to prevent the database from growing too large and creating indexes to optimize performance.

### 19.3.1 Purging old data from tables

A Perl script, `dbPurge.pl`, is provided in the `/bgsys/drivers/ppcfloor/db/schema` directory to purge data from the database. The `dbPurge.pl` script purges environmental data, RAS events, history, performance data, and so on. It is designed to purge data that is older than a specified number of months. In most environments, it is recommended that the script be run daily to prevent the database from growing too large. The recommended data retention period is three months. In a large system environment, purging data older than one month might be necessary. The `dbPurge.pl` script is the only supported method of deleting old database information. Using the `dbPurge.pl` command without options will default to deleting all rows of data that are beyond three months old. The following options can be used to change the default behavior:

- properties <properties-file-name>**  
Defines an alternate `bg.properties` file. The default file, `/bgsys/local/etc/bg.properties` or the location specified by `BG_PROPERTIES_FILE` environment variable if defined is used if the option is not specified.
- months, -m <number\_of\_months>**  
Number of months that determines which rows to delete. Rows older than the number of months passed in from the current date are purged from the tables.
- v**  
Does not actually purge anything. Instead it just reports on which tables it would purge, and how many rows it would purge in each.
- env**  
Only purges rows from the `TBGQ*ENVIRONMENT` and `TBGQ*TEMP` tables.
- hist**  
Only purges rows from the `TBGQ*_HISTORY` tables.
- event**  
Only purges rows from the `TBGQ*EVENTLOG` tables.
- perf**  
Only purges rows from the `TBGQ*PERF` tables.
- diags**  
Only purges rows from the `TBGQDIAG*` tables.
- all**  
Purges rows from the `ENVIRONMENT`, `PERF`, `DIAGS`, and `HISTORY` tables along with the `EVENTLOG` tables.
- q <text>**  
Further qualify the table(s) to purge using text that must be part of the table name.
- h, --help**  
The command help text.

One method of implementing the data purge is to execute a simple wrapper script on the service node, using cron, every night at midnight. Example 19-2 shows `bgqadmin`'s crontab information that initiates the `dbPurge.sh` script.

*Example 19-2 bgqadmin's crontab entry to run dbPurge.sh*

---

```
$crontab -l
0 0 * * * /bgsys/local/etc/dbPurge.sh
```

---

Example 19-3 on page 214 shows the contents of the calling script, `dbPurge.sh`, that was placed in the `/bgsys/local/etc` directory. `dbPurge.sh` is not included in the Blue Gene/Q software. The contents of Example 19-3 on page 214 can be used to create the script on the service node. The example script overrides the default three-month interval and sets it to one month.

*Example 19-3 Contents of simple dbPurge.sh*


---

```
#!/bin/bash
source /dbhome/bgqsysdb/sql1lib/db2profile
/bgsys/drivers/ppcfloor/db/schema/dbPurge.pl --all -m 1
```

---

The calling script can be modified to suit individual environments. The script shown in Example 19-4 is set up to delete RAS events older than three months, and environmentals older than two months.

*Example 19-4 dbPurge.sh script*


---

```
#!/bin/bash
source /dbhome/bgqsysdb/sql1lib/db2profile
/bgsys/drivers/ppcfloor/db/schema/dbPurge.pl --event -m 3
/bgsys/drivers/ppcfloor/db/schema/dbPurge.pl --env -m 2
```

---

## 19.3.2 Optimizing with indexes

The following steps can be used to automatically create indexes that will improve database performance on a Blue Gene/Q service node. The steps might take several hours to complete. The steps can be done during normal operation of the machine. The procedures must be done by someone with a general familiarity with using DB2 on a Blue Gene/Q service node, with the ability to connect to the BGDB0 database, and who has the password for the bgqsysdb userid.

This procedure is optional. If the machine is running normally, and there are no issues with database performance, such as slow response from Blue Gene Navigator, or other database operations, there is no need to use this procedure. This procedure must only be used if there is some evidence of database performance problems.

Before starting, verify that DB2 is set to do automatic maintenance of database statistics. The procedures in this document rely on the statistics maintained by DB2 to make accurate recommendations on index creation.

1. Display the current maintenance settings with the following command:

```
db2 get db config for bgdb0 | grep AUTO_
```

The command should return results similar to those in Example 19-5.

*Example 19-5 DB2 maintenance settings*


---

```
Auto deletion of recovery objects (AUTO_DEL_REC_OBJ) = OFF
Automatic maintenance (AUTO_MAINT) = ON
Automatic database backup (AUTO_DB_BACKUP) = OFF
Automatic table maintenance (AUTO_TBL_MAINT) = ON
Automatic runstats (AUTO_RUNSTATS) = ON
Automatic statement(AUTO_STMT_STATS) = OFF
Automatic statistics profiling (AUTO_STATS_PROF) = OFF
Automatic profile updates (AUTO_PROF_UPD) = OFF
Automatic reorganization (AUTO_REORG) = OFF
Auto-Revalidation (AUTO_REVAL) = DISABLED
```

---

2. Ensure that the values Automatic maintenance, Automatic table maintenance, and Automatic runstats are set to ON. If they are not ON, they can be turned on using this command:

```
db2 update db cfg for bgdb0 using auto_runstats ON auto_tbl_maint ON auto_maint ON
```



If the settings were not ON, and had to be turned on, then you should run for at least a week to accumulate accurate statistics before running the rest of this procedure. You will get improved database performance on your service node by having indexes that are best suited to the usage patterns on your machine.

If these automatic maintenance features are not present in your installed version of DB2, you must upgrade to the latest version of DB2.

The main procedure is described in the following steps. When the steps are completed, the procedure is done. This procedure can be run again every few months in case there are changes in usage patterns. Run the following steps as userid bgqsysdb.

1. Set your userid using the command:

```
su bgqsysdb
```

Supply the password when prompted.

2. Create a new directory on your service node. Using a text editor create three files, *extract.sed*, *extract\_idx.sed*, and *parse.awk* in the new directory. The contents of each file are shown in the following examples.

- i. Copy the contents of Example 19-6 into the *extract.sed* file.

*Example 19-6 extract.sed file contents*

---

```
0,/Dynamic SQL Statements:/ d
/Address AnchID StmtUID NumEnv NumVar NumRef NumExe Text/ d
/Dynamic SQL Environments:/ { g; q; }
```

---

- i. Copy the contents of Example 19-7 into the *extract\_idx.sed* file.

*Example 19-7 extract\_idx.sed file contents*

---

```
0,/-- LIST OF RECOMMENDED INDEXES/ d
/ADVISOR DETAILED XML OUTPUT/ { h ; q; }
```

---

- i. Copy the contents of Example 19-8 into the *parse.awk* file.

*Example 19-8 parse.awk file contents*

---

```
BEGIN { RS="\n0x"; IGNORECASE=1; }
{
d = $8;
i = 9;
while (i <= NF) { d = d " " $i; ++i; }
if ($8 == "select" || $8 == "SELECT") {
print "-- #SET FREQUENCY " $7;
print d;
print ";";
}
}
```

---

3. Extract SQL statements from the dynamic statement cache using the following command:

```
db2pd -d bgdb0 -dynamic > ./db2pd.out
```

4. Convert the file into the format required by the DB2 Design Advisor using the following command:

```
sed -f extract.sed < ./db2pd.out | awk -f parse.awk | sed "s/block
d/blockid/g" > ./db2advis.in
```

- Run the DB2 Design Advisor with this command:

```
db2advvis -d bgdb0 -i ./db2advvis.in > db2advvis.out
```

**Note:** This command might run for several hours depending on the service node, and the amount of DB2 activity. If this step fails due to missing explain tables, run:

```
db2 -tf EXPLAIN.DDL
```

from the ~bgqsysdb/sqllib/misc/ directory.

If db2advvis still fails with errors about the explain tables, drop the objects created by EXPLAIN.DDL because they might be out of date. Then run db2 -tf EXPLAIN.DDL again.

- Extract the list of CREATE INDEX statements from the DB2 Design Advisor output by running the following command:

```
cat db2advvis.out | sed -f extract_idx.sed > ./create_index.sql
```

- Create the recommended indexes:

```
db2 connect to bgdb0
```

```
db2 -tvf ./create_index.sql
```

If for any reason, you want to undo the index creation, the following commands will drop the indexes created in step 5.

```
cat db2advvis.out | grep "CREATE.* INDEX" | sed "s/ON.*//" | sed "s/*BGQSYSDB\"./DROP INDEX BGQSYSDB./" > drop_index.sql
```

```
db2 connect to bgdb0
```

```
db2 -vf ./drop_index.sql
```

```
db2 commit work
```

### 19.3.3 Reorganizing tables

Reorganizing tables can reduce the amount of disk space used. This operation should be done periodically. In Figure 19-1 on page 217, the **dbReorg.sh** script is added to the cron.weekly crontab file to run once per week.

```
bgqsn@bqadmin:cat /etc/cron.weekly/dbReorg.sh
#!/bin/bash

. ~bgqsysdb/sqllib/db2profile

db2 connect to bgdb0

db2 "select tabname from syscat.tables where tabname like 'TBGQ%' and
tabschema='BGQSYSDB'" | grep TBGQ | awk '{print $1}' > /tmp/bgdb0.tables

for i in `cat /tmp/bgdb0.tables`
do
 echo "Reorganizing Table - $i"
 db2 "reorg indexes all for table $i allow write access"
 db2 "reorg table $i"
done

bgqsn@bqadmin:
```

Figure 19-1 Sample script for reorganizing tables





# Logging

Control System logs diagnose software, hardware, and configuration problems. This chapter provides information about Apache log4j and log4cxx, log file format, log merge utility, log configuration, and log rotation.

## 20.1 Apache log4j and log4cxx

The Blue Gene/Q Control System programs use an open source library based on Apache log4j for logging. The Apache log4j design provides a large number of features and offers a great deal of customizability. The library can be configured dynamically and includes a facility for initializing the library using a configuration file. In log4j, the central object is a logger. An application typically has several loggers. Each logger has a name. Examples of loggers in the Control System are *ibm.mmcs.mmcs\_server* and *ibm.bgsched allocator.Allocator*. Another attribute of a logger is its level, which is the cut-off for the message level (or severity) that will be logged. The message levels in log4j are TRACE, DEBUG, INFO, WARN, ERROR, and FATAL.

Loggers form a hierarchy, where a child logger can inherit or override the message level of its parent logger. For example, the two Control System loggers used in the previous paragraph are both descendants of the “ibm” logger. Because child loggers inherit their parent logger’s attributes, unless overridden, an administrator can quickly and easily configure all of the loggers in an application. It is often also useful when debugging to override the level for a single logger.

Each logger is also associated with an appender. The appender determines where log records go. The appenders provided by the log4j library can send log records to the program’s standard output, to a file, to syslog, and various other locations. Text-file based appenders have a layout that formats the log record for output. By setting the layout consistently, the log records have a standard format.

The log4j design was ported to several programming languages. The Control System is written in C++ and uses the log4cxx library.

## 20.2 Log file format

The Control System logs use a common format for log records. Each log record begins with the time stamp in standard ISO 8601 format. An example time stamp found in a log file is “2010-10-01 14:38.627”. After the time stamp, the log record contains the message’s level (TRACE, DEBUG, INFO, WARN, ERROR, or FATAL). After the level is the thread ID, which is useful for debugging multi-threaded applications. Next, the name of the logger is displayed, and finally the message that is being logged.

The following syntax is the standard format used for log message on Blue Gene/Q using the log4j formatting language:

```
%d{yyyy-MM-dd HH:mm:ss.SSS} (%-5p) [%t] %c: %m%n
```

Some Control System applications take advantage of a feature of the log4j design called Mapped Diagnostic Context (MDC) to include context for a log record, such as the name of the user or a client ID when the application can process multiple clients.

Example 20-1 contains lines from the MMCS server log file when it starts and a later message indicating an error.

*Example 20-1 Sample MMCS log file*

---

```
2011-10-01 09:11:59.273 (INFO) [0xffff7bbdb10] ibm.mmcs.mmcs_server: MMCS[18618]:
starting: DRV400_2011 (revision 47346) Sep 29 2011 04:39:34
2011-10-01 09:11:59.274 (INFO) [0xffff7bbdb10] ibm.mmcs.mmcs_server: Startup parameters:
mmcs_server
```

```
2011-10-01 09:11:59.334 (INFO) [0xffff7bbdb10] ibm.mmcs.MMCSConsolePort: Attempting to
bind and listen on 2 port pairs.
2011-10-01 09:11:59.335 (INFO) [0xffff7bbdb10] ibm.mmcs.mmcs_server: MMCS: console port
open.
2011-10-01 09:37:58.118 (ERROR) [0xffff561ff0c0] ibm.mmcs.MMCSCommandProcessor:
allocate_block: FAIL;create_block: resources are unavailable - CABLE: D_R00-MID_R00-M0
```

---

## 20.3 Log merge utility

The Control System writes several log files in `/bgsys/logs/BGQ` that are generated by the different servers, the runjob muxes running on the front end nodes, and Common I/O Services (CIOS) running on the I/O nodes. When debugging a problem, developers and service personnel often examine log files from multiple servers for clues as to the source of the problem. Often they want to see the log records from multiple servers interleaved in the same file to examine what has happened on the servers. The Blue Gene/Q log merge utility can create a merged log file.

The `log_merge` reads Control System log files and merges them, displaying log entries from different log files in order of the time stamp and prefixed with the name of the file the log record came from. Blue Gene/Q log files use a common format for log entries that allows them to be merged in this way.

The output of `log_merge` is the log entries from the log files, with each entry prefixed with the filename or the process name. Use the `--map` (or `-m`) option to print the process name rather than the filename.

The input files can be specified on the command line or read from standard input. If no files are given then the log files used are the ones in the default location, `/bgsys/logs/BGQ`, or the value for the log directory in the Blue Gene/Q configuration file (`bg.properties`). If a directory is given then `log_merge` will use every file in the directory. It does not recurse into subdirectories of the given directory.

If `-`, `-0`, or `--null` is given on the command line, `log_merge` reads file or directory names from standard input. The filenames must be separated by a newline character unless `-0` or `--null` is given on the command line. The filenames must be separated by a null character if `-0` or `--null` is used. If you use the `find` command to generate the list of files, use `-print0` to generate a list of filenames separated by a null character.

The `log_merge` works in one of two modes: historical or live. The default mode is historical. In historical mode, `log_merge` reads log entries within the interval from the existing log files and merges them into a single stream. The interval is specified using the `--start` and `--end` command-line options.

In live mode, `log_merge` monitors the log files for new output and prints the log entries as they are added. This is useful for monitoring the Blue Gene/Q Control System processes. Use the `--live` option to run `log_merge` in live mode.

The `log_merge` utility is installed to `/bgsys/drivers/ppcfloor/bin/log_merge` and should be in your system path.

### 20.3.1 Configuration

The `log_merge` utility reads the Blue Gene/Q configuration file (`bg.properties`) for configuration. The only configuration option is `"log_dir"` in the `"log_merge"` section. This is the

directory that is used if no arguments are given on the command line. If not present, the default value of “/bgsys/logs/BGQ” is used.

## 20.3.2 Usage

The `log_merge` utility optionally takes a list of file or directory names as the files to merge or the directory containing the files to merge. If no files or directories are given, the command will use the configured directory as described in the Configuration section above. Use a filename of “-” (the dash character) to read file or directory names from standard input. You must have read permission to the file to perform this operation. The syntax of the `log_merge` command is:

```
log_merge [OPTIONS ...]... [FILE or DIRECTORY]
```

The supported `log_merge` command options are as follows:

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>--live</b>           | Live updates.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>-m [ --map ]</b>     | Displays short names for log files.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>--start arg</b>      | Interval start time. See 20.3.3, “Time stamp format” on page 222 for time stamp formatting. This option cannot be used with the <code>--live</code> option.                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>--end arg</b>        | Interval end time. See 20.3.3, “Time stamp format” on page 222 for time stamp formatting. This option cannot be used with the <code>--live</code> option.                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>-0 [ --null ]</b>    | Reads null-terminated file names from standard input.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>--properties arg</b> | Blue Gene/Q configuration file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>--verbose arg</b>    | Sets the logging configuration. Setting these values overrides the logging configuration in the <code>bg.properties</code> file. The <code>--verbose</code> option can be used to set the logging level for the default logger or any additional loggers. The following values are allowed for the level: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, or ALL. For example, if you want to set both the default logger and the <code>ibm.utility</code> logger to the <code>debug</code> value, the following command can be used:<br><b>log_merge --verbose=DEBUG --verbose=ibm.utility=DEBUG</b> |
| <b>-h [ --help ]</b>    | Print command help text.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

## 20.3.3 Time stamp format

The format of time stamps, used in the `--start` and `--end` options, are:

- ▶ YYYY-mm-dd HH:MM::SS.sss
- ▶ YYYY-mm-dd
- ▶ mm-dd
- ▶ HH:MM::SS.sss

In the time stamp, YYYY is the year, mm is the month number, dd is the day, HH is the hour in 24-hour format, MM is the minute, SS is the seconds, and sss is fractions of a second.

When a date is given without a time stamp, the actual time is midnight of that day if it is the `--start` option, or midnight of the next day if it is the `--end` option. For example, to show the log entries for June 7, you can use `--start 06-07 --end 06-07`.



## 20.4 Configuration

The logging configuration is included in the “logging” section of the Blue Gene/Q system configuration file. Because of this, the names of many of the loggers that an administrator wants to change the level for are visible. For debugging purposes, users can also set the logging level for individual loggers by adding them to this file with the new level.

If you alter the format of the log record, the log merge utility will not work because it cannot parse the time stamp from the log record.

Example 20-2 shows the logging configuration in the Blue Gene/Q system configuration file.

*Example 20-2 Logging configuration in the bg.properties file*

---

```
[logging]
log4cxx configuration. For more information, see http://logging.apache.org/log4cxx/
log4j.logger.ibm = INFO, default
ibm logger - Root for logging done by IBM components ('level' progression - TRACE,
DEBUG, INFO, WARN, ERROR, FATAL)
log4j.additivity.ibm = false

log4j.logger.ibm.utility = INFO # Utility functions
log4j.logger.ibm.utility.cxxsockets = WARN # CxxSockets library
log4j.logger.ibm.bgsched = INFO # Scheduler APIs
log4j.logger.ibm.security = INFO # Security
log4j.logger.ibm.database = INFO # Database layer
log4j.logger.ibm.realtime = INFO # Real-time server
log4j.logger.ibm.runjob = WARN # job submission
log4j.logger.ibm.runjob.mux = DEBUG # job submission multiplexer
log4j.logger.ibm.runjob.server = DEBUG # job submission server
log4j.logger.ibm.mc = DEBUG # MC (machine controller)
log4j.logger.ibm.mc.MCServer = DEBUG # mcServer
log4j.logger.ibm.mc.MCServerThread = DEBUG # mcServer
log4j.logger.ibm.mc.MCServerLockNode = DEBUG # mcServer
log4j.logger.ibm.boot = DEBUG # MC boot lib
log4j.logger.ibm.bgqconfig = INFO # BGQ config
log4j.logger.ibm.icon = DEBUG # iCon
log4j.logger.ibm.bgws = INFO # Web Services
log4j.logger.ibm.mmcs = INFO # MMCS server
log4j.logger.ibm.ras = WARN # RAS
log4j.logger.ibm.master = INFO # BGmaster
log4j.logger.ibm.cios = WARN # Common I/O Services

[logging.default]

log4j.appender.default = org.apache.log4j.ConsoleAppender
log4j.appender.default.layout = org.apache.log4j.PatternLayout
log4j.appender.default.layout.ConversionPattern = %d{yyyy-MM-dd HH:mm:ss.SSS} (%-5p) [%t]
%c: %m%n
```

---

## 20.5 Log rotation

There are many log files that exist on a Blue Gene/Q service node, and some can grow large and become difficult to analyze. To resolve this problem, log files can be split into more manageable sizes using the **logrotate** utility that ships with many flavors of Linux, including RedHat. You can learn more about **logrotate** by reading its manual page.

## 20.5.1 Control System logs

The BGmaster component (see Chapter 11, “BGmaster” on page 129) creates log files for each managed process if they do not exist. The file names are in the format \$hostname-alias.log. BGmaster redirects the standard output and error file descriptors of the process to this file. By configuring **logrotate**, you will have a series of compressed log files, each with a date appended to the end, showing when **logrotate** compressed and rotated the file. The log files will not grow much beyond the size specified in the size argument in the **logrotate** configuration file, as shown in Example 20-3.

*Example 20-3 logrotate Control System logs configuration file*

---

```

bgqadmin@bgqsn ~> cat /etc/logrotate.d/bgmaster
/bgsys/logs/BGQ/*-mc_server.log /bgsys/logs/BGQ/*ubnet*.log
/bgsys/logs/BGQ/*-runjob_server.log /bgsys/logs/BGQ/*-runjob_mux*.log
/bgsys/logs/BGQ/*-mmcs_server.log /bgsys/logs/BGQ/*-bgmaster_server.log
/bgsys/logs/BGQ/*-bgagentd.log /bgsys/logs/BGQ/*-bgws_server.log
/bgsys/logs/BGQ/*-realtime_server.log /bgsys/logs/BGQ/teal/*teal*.log {
 rotate 10
 size=100M
 notifempty
 missingok
 copytruncate
 compress
}
bgqadmin@bgqsn ~>

```

---

## 20.5.2 I/O node logs

The MMCS server creates log files for each I/O node in the Blue Gene/Q system. They are by default located in /bgsys/logs/BGQ/location.log where *location* is the I/O node location. It is also useful to rotate these logs to prevent them from growing too large. The **logrotate** configuration file for I/O node logs is shown in Example 20-4.

*Example 20-4 logrotate I/O node logs configuration file*

---

```

bgqadmin@bgqsn ~> cat /etc/logrotate.d/ionodes
/bgsys/logs/BGQ/R*I*J*.log /bgsys/logs/BGQ/Q*I*J*.log {
 rotate 10
 size=10M
 notifempty
 missingok
 copytruncate
 compress
}
bgqadmin@bgqsn ~>

```

---

## 20.5.3 Subnet service node logs

Every subnet service node (see Chapter 17, “Distributed Control System” on page 193) also needs a **logrotate** configuration file defined to manage the size of log files. The **logrotate** configuration file for subnet service node logs is shown in Example 20-5 on page 225.

---

*Example 20-5 logrotate subnet service node logs configuration file*

---

```
bgqadmin@bgqssn6 ~> cat /etc/logrotate.d/bgmaster
/bgsys/logs/BGQ/*/*ubnet*.log /bgsys/logs/BGQ/*/*-bgagentd.log {
 rotate 10
 size=100M
 notifempty
 missingok
 copytruncate
 compress
}
bgqadmin@bgqssn6 ~>
```

---

## 20.5.4 Log maintenance

The majority of the logs created by Control System daemons are rotated by the **logrotate** utility. Other logs created in subdirectories of `/bgsys/logs/BGQ` are not rotated or purged by **logrotate**. To manage these logs, add a script to the `cron.daily` directory so that a daily job will run that cleans up the logs, as shown in Example 20-6.

---

*Example 20-6 cron.daily script to clean up old logs*

---

```
bgqadmin@bgqsn ~> cat /etc/cron.daily/clean_bgsys_logs
#!/bin/bash
Clean old log files
find /bgsys/logs/BGQ -type f -mtime +30 -exec rm {} \;
find /bgsys/logs/BGQ/diags/bringup -type f -name "*EdramChargePump*.log" -ctime +7 -exec rm
-f "{}" \;
bgqadmin@bgqsn ~>
```

---





## The bg.properties file

The properties file is the single central location for all Blue Gene/Q software configuration information. The contents of the file are used to configure database connection information, configure and control logging behavior, and enable the configuration of every component present in the Blue Gene/Q software stack.

## 21.1 The bg.properties file overview

The bg.properties file is located in the /bgsys/local/etc directory. The file contains no plain text passwords and must be readable by all users. The format of the file is similar to an INI file.

Characteristics and limitations of the file are:

- ▶ Section names are enclosed in square brackets.
- ▶ Subsections are also in square brackets but have one or more periods in the name.
- ▶ Comments can start anywhere in a line and are denoted by a single pound '#' character.
- ▶ Duplicate sections are invalid.
- ▶ Duplicate keys within a section are invalid.

Example 21-1 shows the database configuration and the runjob server sections of the bg.properties file.

*Example 21-1 bg.properties file format*

---

```
Blue Gene configuration file.

[database]

name=BGDBO
 # The name of the database containing the Blue Gene schema.

#user=bgqsysdb
 # The user to connect to the database as.
 # This value is optional. If not set, then the current user is used.

#password=xxxxxxx
 # The password to use when connecting to the database.
 # This value is required if the user is set and ignored if it's not.

schema_name=bgqsysdb
 # The database schema containing the Blue Gene tables.
 # This value is optional. If not set, the default schema is used.

computeRackRows=1
computeRackColumns=4
 # Used by dbPopulate to populate the schema.

...

[runjob.server]

thread_pool_size = auto
 # Number of threads to run in the server.
 # Allowed values are integers greater than 0 or "auto".
 # "auto" means the runjob_server will pick the number of threads to start.
 # The default is auto.

mux_listen_ports = localhost:25510,172.20.3.1:25510
command_listen_ports = localhost:24510,172.20.3.1:24510
 # The ports that the runjob_server will listen for connections on.
 # The format is a comma-separated list of host:port pairs. host is an IP
 # address or host name. IP addresses containing colons must be enclosed in
```

```
[]. Link local ipv6 addresses must include the interface name appended
after a % character. port is a port number or service name.
The host and port are optional. If the host is not specified the server
will listen on any interface. If the port is not specified the default
port will be used (25510 and 24510).

io_connection_interval_increment = 5
Number of seconds to increment a progressive timeout when
retrying connection attempts to the CIOS (common I/O services) daemons.
This value can be updated by the runjob_server_refresh_config command.
io_connection_interval_max = 120
Maximum number of seconds to wait between connection attempts to the CIOS
daemons. This value can be updated by the runjob_server_refresh_config
command.

performance_counter_interval = 30
Number of seconds to wait between inserting performance counter
statistics into the database for persistent storage. The default is 30
seconds if this value is not specified. This value can be updated by the
runjob_server_refresh_config command.

connection_pool_size = 20
The maximum number of connections held in the runjob_server database
connection pool.
If not specified, the maximum is 20.

extra_connection_parameters = Debug=True;
Extra parameters to pass on the connection.
Format is NAME=VALUE pairs separated by ;.
Optional, default is no extra parameters.

output_throttle = 128
The maximum throughput per I/O connection for standard output and
standard error. The units are kilobytes per second. If not specified, the
value is 128. This value can be updated by the
runjob_server_refresh_config command.

output_delay = 500
The amount of milliseconds to delay reading from a stdio daemon running
on an I/O node if it has hit the output_throttle maximum throughput
value. If not specified, this value is 500.
This value can be updated by the runjob_server_refresh_config command.

control_action_heartbeat = 60
The number of seconds to wait between checking for progress of a job that
has been terminated due to a RAS event with a control action. If not
specified, this value is 60.
This value can be updated by the runjob_server_refresh_config command.

...
```

---

## 21.2 Property file validation

The server configuration sections of the bg.properties file can be validated for correctness using the **properties\_validate** command. The command ensures that the configuration options that must be present are there and that all dependencies are met. Any failures detected by the command can cause servers to fail on start up. The accepted arguments for the **properties\_validate** command are:

- properties** This required argument is the fully qualified file name to validate (for example, /bgsys/local/etc/bg.properties).
- server** This option allows specification of a single server. Valid values are bgmaster\_server and mmcs\_server. If this argument is not specified, the default is for the configuration of all supported servers (for example, bgmaster\_server and mmcs\_server) to be checked.
- dynamic** Specifying this option provides extended analysis of runtime options, such as port configurations and file permissions. Dynamic analysis must only be run on the system that is running the processes under test. If you are testing BGmaster configuration, run the **properties\_validate** command on the primary service node.
- help** Prints the help text.





## The Coreprocessor tool

This chapter describes the Blue Gene/Q Coreprocessor tool, which is a basic parallel debugger that enables debugging 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 has 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.

## 22.1 Usage and dependencies

You can use the Coreprocessor tool for the following purposes:

- ▶ To examine compute node core files. For performance and disk space reasons, core files are simple text files, and their format is not understood by all debuggers. The Coreprocessor tool helps determine which node is acting in a suspicious manner and caused the block to dump.
- ▶ When a standard debugger cannot connect to the compute nodes. Other debuggers are not useful when a node is non-responsive. 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.

**Note:** Coreprocessor does not support debugging sub-block jobs.

Using the Coreprocessor tool provides the following advantages:

- ▶ The operating system can be completely dead and can still handle debug
- ▶ Quick isolation of nodes that are abnormal
- ▶ Scales to a full Blue Gene/Q 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 Coreprocessor tool uses the following Perl modules:

- ▶ Cwd
- ▶ Compress::Zlib
- ▶ IO::Socket
- ▶ OpenGL (for 3D Map function only)
- ▶ Tk

**Note:** Tk is not usually installed with the operating system. It can be installed by running the following command:

```
sudo perl -MCPAN -e "install Tk" \;
```

Answer *no* at the prompt to let it autoconfigure.

## 22.2 Starting the Coreprocessor tool

The Coreprocessor tool is initiated from a Virtual Network Computing (VNC) session. From the command line (VNC), enter the following command:

```
/bgsys/drivers/ppcfloor/coreprocessor/bin/coreprocessor.pl
```

The tool does not require parameters to start. Table 22-1 on page 233 provides a list of options that you can use when starting the tool.

Table 22-1 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:<br>-b=/bguser/username/hello_world.elf:/bgsys/drivers/ppcfloor/boot/cnk                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| -c         | Specifies the path to the directory containing existing core files. Example:<br>-c=/bgusr/username/<br>You can use the following options:<br>-mincore Specifies the lowest numbered core file to load. The default is 0.<br>Example: -mincore=75<br>-maxcore Specifies the highest numbered core file to load. The default is 131072. Example: -maxcore=65536<br>-coreprefix Specifies the filename prefix for lightweight core files. The default is core. Example: -coreprefix=core                                                                                                                                                                                                                      |
| -a         | Connects to MC server and attaches to the specified hardware. The format is a Perl regular expression. Example:<br>-a=R00-M0-N00-J..<br>You can use the following options:<br>-user The user ID of the specified hardware. Example:<br>-user=bgqadmin<br>-numbadframes Number of the topmost stack frames to remove from collection. This is useful if the application under debug is doing some custom assembly. Under these conditions the application might not strictly adhere to the PowerPC ABI governing stack frame usage. Example:<br>-numbadframes=1                                                                                                                                             |
| -s         | Loads the specified Coreprocessor Snapshot file for post-failure analysis. Example:<br>-s=/bgusr/username/linpack.snapshot                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| -templates | Specifies the directory for saved memory templates. Example:<br>-templates=/bgusr/username/mytemplates/.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| -nogui     | GUI-less mode. Coreprocessor tool processes the data and terminates. Example:<br>-nogui<br>You can use these options:<br>-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=DCR<br>-register Specifies the PPC or DCR register when using the PPC or DCR modes. Example: -register=GPR3<br>-snapshot Fetches failure data and saves to snapshot file. Example:<br>-snapshot=/tmp/hang_issue1234<br>-flightlog Suggests which nodes to collect flight log data from based on unique IARs. Example: -flightlog |
| -j         | Connects to a running job through the scalable debugger interface. Example:<br>-j=57132                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| -h         | Displays help text. Example: -h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

Figure 22-1 on page 234 shows the graphical user interface (GUI) Coreprocessor tool started in a VNC session.

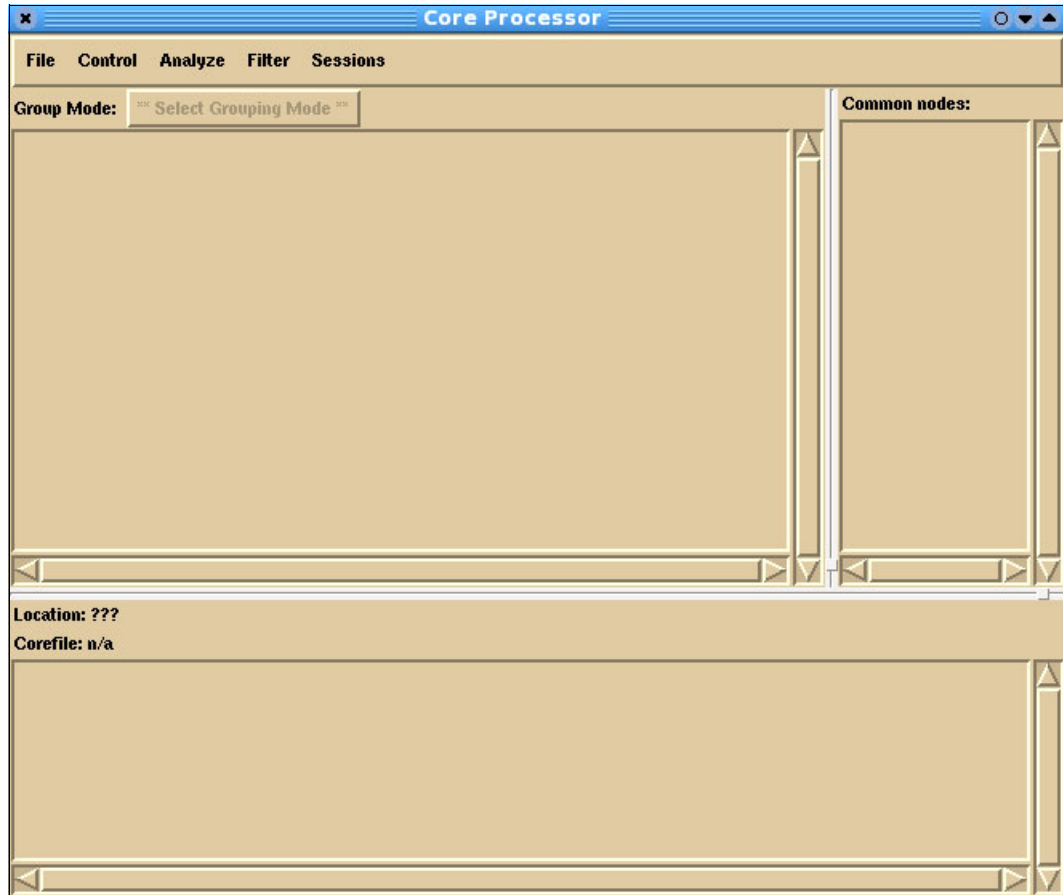


Figure 22-1 Coreprocessor GUI

## 22.3 Debugging live compute node problems

When debugging an unresponsive job, 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, and then select **File Attach To Block**. The window in Figure 22-2 opens.



Figure 22-2 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 hardware. This is a regular expression that matches the locations used by the block. For example, R00-M0-N00-J.
  - 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 example:  

```
/bgsys/drivers/ppcfloor/boot/cnk:/bgusr/bgquser1/mpi_hang_test.bg
```
  - User name or current owner of the hardware.
3. Click **Attach**. At this point, you have not yet affected the state of the processors.
  4. Back at the main window, 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 that the last stack frame is grouped based on the function name (not the IAR), as shown in Figure 22-3. This mode is normally the most useful mode for debug.

Notice the text in the main pane. The numbers *2176*, *64*, *32*, and *so on*. In parenthesis are the number of nodes in the block that share that function call in the call stack. The number in parenthesis always indicates the number of nodes that share whatever attribute is displayed on the line (the function name in this case).

```

Group Mode: Stack Traceback (condensed)
0 :Node (2176)
1 : KERNEL_CHECKSUM (64)
2 : __libc_start_main (32)
3 : .generic_start_main (32)
4 : .main (32)
5 : .mpi_hang() (32)
6 : .PMPI_Recv (32)
7 : .PAMI_Context_trylock_advancev (32)
8 : .PAMI_Context_trylock_advancev (8)
8 : .PAMI::Interface::Context<PAMI::Context>::adv
9 : .PAMI::Device::Interface::BaseDevice<PAMI
9 : 0000001d3fffb570 (8)
10: .PAMI::Device::Interface::BaseDevice<
9 : .PAMI::Interface::Context<PAMI::Context>:
2 : 0000000041006e64 (32)
3 : 0000000041006b68 (32)
4 : 00000000410017fc (32)
5 : 000000004100180c (32)
1 : .Kernel_Finish (96)
2 : .Kernel_Begin (32)

```

Location: ???  
Corefile: n/a

Figure 22-3 Stack traceback (condensed)

## Stack traceback (detailed)

In Stack Traceback, detailed data from all nodes are captured, as shown in Figure 22-4. The unique instruction addresses per stack frame are grouped and displayed. The IAR at each stack frame is also displayed.

```

Group Mode: Stack Traceback (detailed)
0 : (IAR=Node)Node (2176)
1 : (IAR=0x0000000000000000) KERNEL_CHECKSUM (64)
2 : (IAR=0x00000000013147b4) .__libc_start_main (32)
3 : (IAR=0x00000000013144b8) .generic_start_main (32)
4 : (IAR=0x0000000001000644) .main (32)
5 : (IAR=0x000000000100056c) .mpi_hang() (32)
6 : (IAR=0x00000000010093e0) .PMPI_Recv (32)
7 : (IAR=0x0000000001090ca8) .PAMI_Context_trylock_advancev (32)
8 : (IAR=0x0000000001090c68) .PAMI_Context_trylock_advancev (1)
8 : (IAR=0x0000000001090c70) .PAMI_Context_trylock_advancev (1)
8 : (IAR=0x0000000001090c9c) .PAMI_Context_trylock_advancev (1)
8 : (IAR=0x0000000001090c7c) .PAMI_Context_trylock_advancev (2)
8 : (IAR=0x0000000001090ce0) .PAMI_Context_trylock_advancev (3)
8 : (IAR=0x000000000118c270) .PAMI::Interface::Context<PAMI::Conte
9 : (IAR=0x000000000118bab0) .PAMI::Device::Interface::BaseDev
9 : (IAR=0x000000000118bac0) .PAMI::Device::Interface::BaseDev
9 : (IAR=0x000000000118bad8) .PAMI::Device::Interface::BaseDev
9 : (IAR=0x000000000118b58) .PAMI::Device::Interface::BaseDev
9 : (IAR=0x000000000118b43c) .PAMI::Interface::Context<PAMI::C
9 : (IAR=0x000000000118bddc) .PAMI::Interface::Context<PAMI::C

Location: ???
Corefile: n/a

```

Figure 22-4 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 that contain the same IAR. The stack data fetched for that one node is then applied to all nodes with the same IAR. Figure 22-5 shows an example of survey mode.

```

Group Mode: Stack Traceback (survey)
0 : Node (2176)
1 : KERNEL_CHECKSUM (64)
2 : .__libc_start_main (32)
3 : .generic_start_main (32)
4 : .main (32)
5 : .mpi_hang() (32)
6 : .PMPI_Recv (32)
7 : .PAMI_Context_trylock_advancev (32)
8 : .PAMI_Context_trylock_advancev (8)
8 : .PAMI::Interface::Context<PAMI::Context>::advance(unsigned lo
9 : .PAMI::Device::Interface::BaseDevice<PAMI::Device::MU::Cov
9 : 00000001d3fffb570 (8)
10: .PAMI::Device::Interface::BaseDevice<PAMI::Device::MU:
9 : .PAMI::Interface::Context<PAMI::Context>::advance(unsigned
2 : 0000000041006e64 (32)
3 : 0000000041006b68 (32)
4 : 00000000410017fc (32)
5 : 000000004100180c (32)
1 : .Kernel_Finish (64)
2 : .Kernel_Begin (32)

Location: ???
Corefile: n/a

```

Figure 22-5 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, as shown in Figure 22-6.

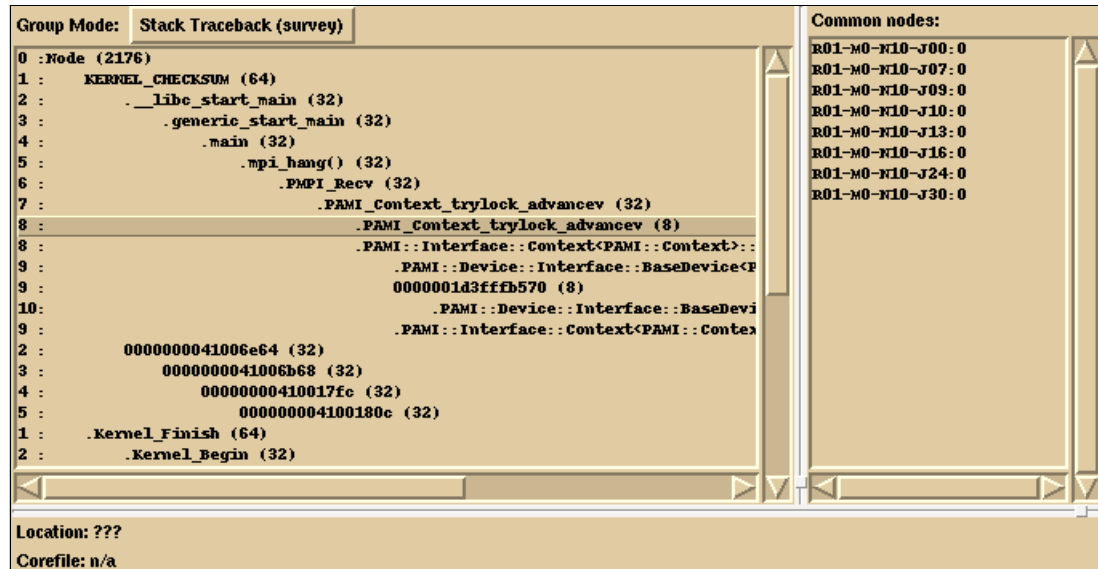


Figure 22-6 Stack traceback common nodes

- ▶ 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.

## 22.4 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 Session 1 and the second one is Session 2. The snapshot is just what the name implies—a picture of the debug session at a particular point. Notice that you cannot 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.

## 22.5 Debugging live I/O node problems

It is possible to debug the I/O nodes and 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.

## 22.6 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 22-7 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: 0x01093fac (1)
17: 0x00000000 (1)
5 : _wordcopy_bwd_dest_aligned (95)
6 : 0x010b92c4 (95)
7 : 0x010dada4 (95)

```

Figure 22-7 Core file condensed stack trace



Figure 22-8 shows the detailed version of the same trace.

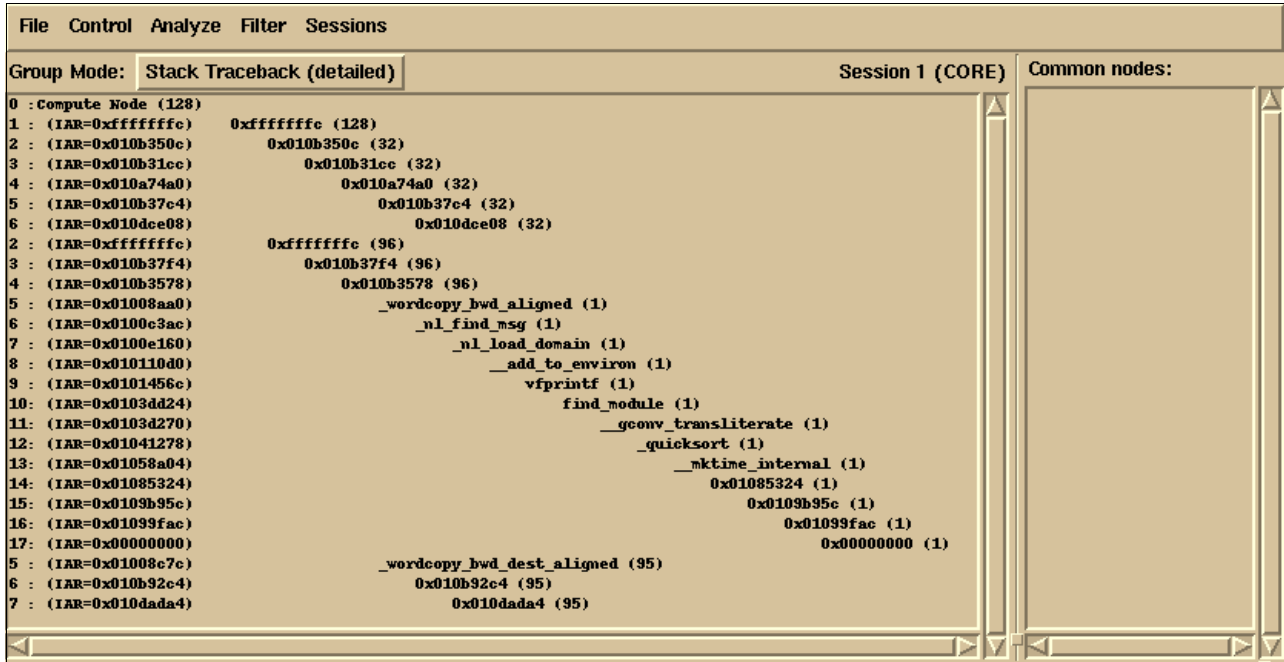


Figure 22-8 Core file detailed stack trace

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, as shown in Figure 22-9 on page 240. 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.

| File Control Analyze Filter Sessions                                                                          |                                                    | Session 1 (CORE) | Common nodes:                               |
|---------------------------------------------------------------------------------------------------------------|----------------------------------------------------|------------------|---------------------------------------------|
| Group Mode: Stack Traceback (detailed)                                                                        |                                                    |                  | disasm 0x0100c3ac<br>core_T6ID_155_Thread_0 |
| 0                                                                                                             | : Compute Node (128)                               |                  |                                             |
| 1                                                                                                             | : (IAR=0xffffffff) 0xffffffff (128)                |                  |                                             |
| 2                                                                                                             | : (IAR=0x010b350c) 0x010b350c (32)                 |                  |                                             |
| 3                                                                                                             | : (IAR=0x010b31cc) 0x010b31cc (32)                 |                  |                                             |
| 4                                                                                                             | : (IAR=0x010a74a0) 0x010a74a0 (32)                 |                  |                                             |
| 5                                                                                                             | : (IAR=0x010b37c4) 0x010b37c4 (32)                 |                  |                                             |
| 6                                                                                                             | : (IAR=0x010dce08) 0x010dce08 (32)                 |                  |                                             |
| 2                                                                                                             | : (IAR=0xffffffff) 0xffffffff (96)                 |                  |                                             |
| 3                                                                                                             | : (IAR=0x010b37f4) 0x010b37f4 (96)                 |                  |                                             |
| 4                                                                                                             | : (IAR=0x010b3578) 0x010b3578 (96)                 |                  |                                             |
| 5                                                                                                             | : (IAR=0x01008aa0) _wordcopy_bwd_aligned (1)       |                  |                                             |
| 6                                                                                                             | : (IAR=0x0100c3ac) _nl_find_msg (1)                |                  |                                             |
| 7                                                                                                             | : (IAR=0x0100e160) _nl_load_domain (1)             |                  |                                             |
| 8                                                                                                             | : (IAR=0x010110d0) __add_to_envIRON (1)            |                  |                                             |
| 9                                                                                                             | : (IAR=0x0101456c) vfprintf (1)                    |                  |                                             |
| 10                                                                                                            | : (IAR=0x0103dd24) find_module (1)                 |                  |                                             |
| 11                                                                                                            | : (IAR=0x0103d270) __gconv_transliterate (1)       |                  |                                             |
| 12                                                                                                            | : (IAR=0x01041278) _quicksort (1)                  |                  |                                             |
| 13                                                                                                            | : (IAR=0x01058a04) __mktime_internal (1)           |                  |                                             |
| 14                                                                                                            | : (IAR=0x01085324) 0x01085324 (1)                  |                  |                                             |
| 15                                                                                                            | : (IAR=0x0109b95c) 0x0109b95c (1)                  |                  |                                             |
| 16                                                                                                            | : (IAR=0x01099fac) 0x01099fac (1)                  |                  |                                             |
| 17                                                                                                            | : (IAR=0x00000000) 0x00000000 (1)                  |                  |                                             |
| 5                                                                                                             | : (IAR=0x01008c7c) _wordcopy_bwd_dest_aligned (95) |                  |                                             |
| 6                                                                                                             | : (IAR=0x010b92c4) 0x010b92c4 (95)                 |                  |                                             |
| 7                                                                                                             | : (IAR=0x010dada4) 0x010dada4 (95)                 |                  |                                             |
| Location: /bglhome/usr5/bgbuild/DRV050_2007-070130P-SLES9-DD1-GNU/ppc/bgp/gnu/glibc-2.3.6/nl/dcigettext.c:789 |                                                    |                  |                                             |
| Corefile: n/a                                                                                                 |                                                    |                  |                                             |

Figure 22-9 Core files common nodes

**A**

# Hardware location naming conventions

This appendix provides an overview of how the Blue Gene/Q hardware locations are assigned. This naming convention is used consistently throughout both the hardware and software.

## Letter designation reference

The following letter designations are used for naming Blue Gene/Q hardware component locations:

- A = PCI Adapter Card
- B = Bulk Power Supply
- C = Compute Card Core
- D = Direct Current Assembly (DCA) Card
- E = Fiber Optic Ribbon
- F = Fan
- H = Fan Assembly
- I = I/O Drawer
- J = Compute Card
- K = Clock Card
- L = Coolant Monitor
- M = Midplane
- N = Node Board
- O = Optical Module
- P = Power Module
- Q = I/O rack
- R = Compute Rack
- S = Service Card
- T = Node/IO Fiber Adapter Port
- U = Link/Compute Module

# Hardware location naming convention

Figure A-1 shows the Blue Gene/Q naming convention used when assigning locations for racks, power supplies, midplanes, and service cards.

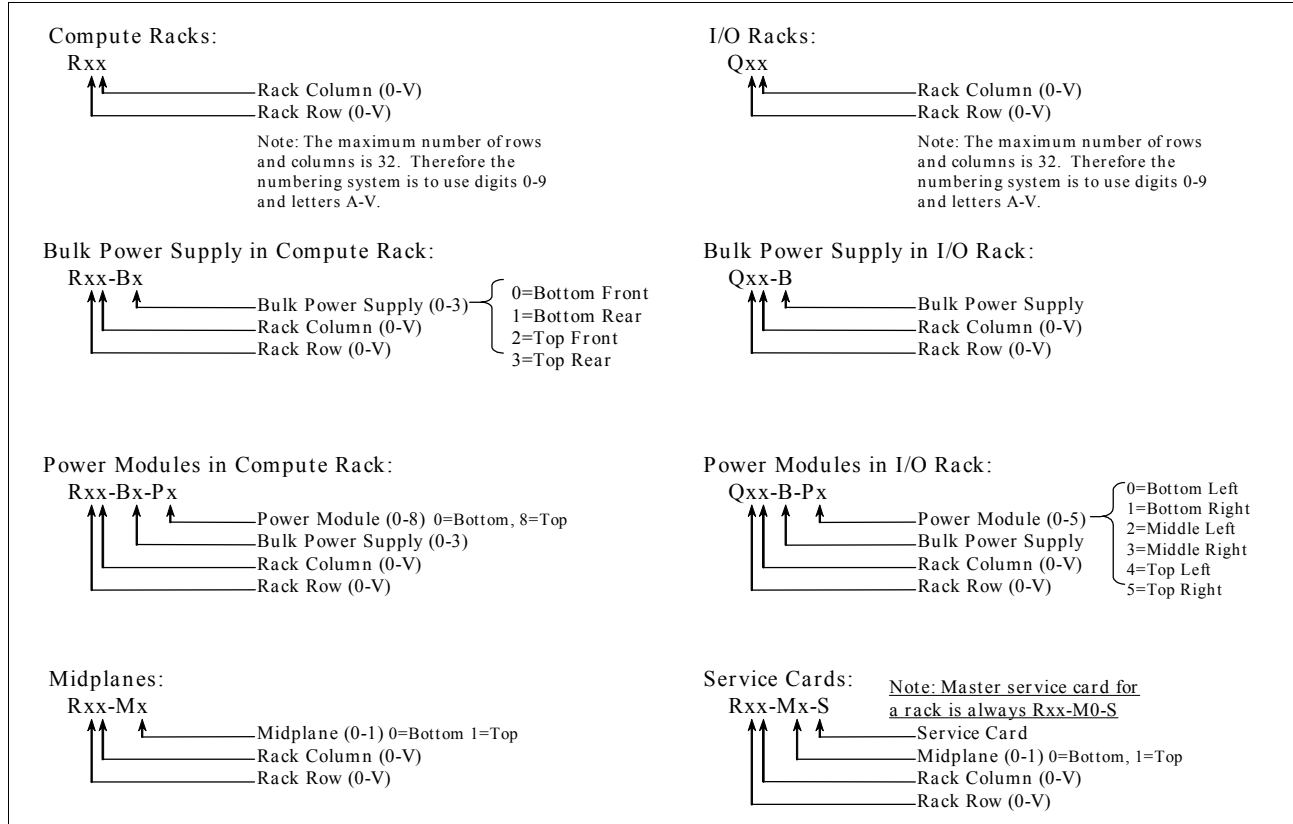


Figure A-1 Hardware naming convention for racks, power supplies, midplanes and service cards

Figure A-2 shows the Blue Gene/Q naming convention used when assigning locations for clock cards, I/O drawers, node boards, and compute cards.

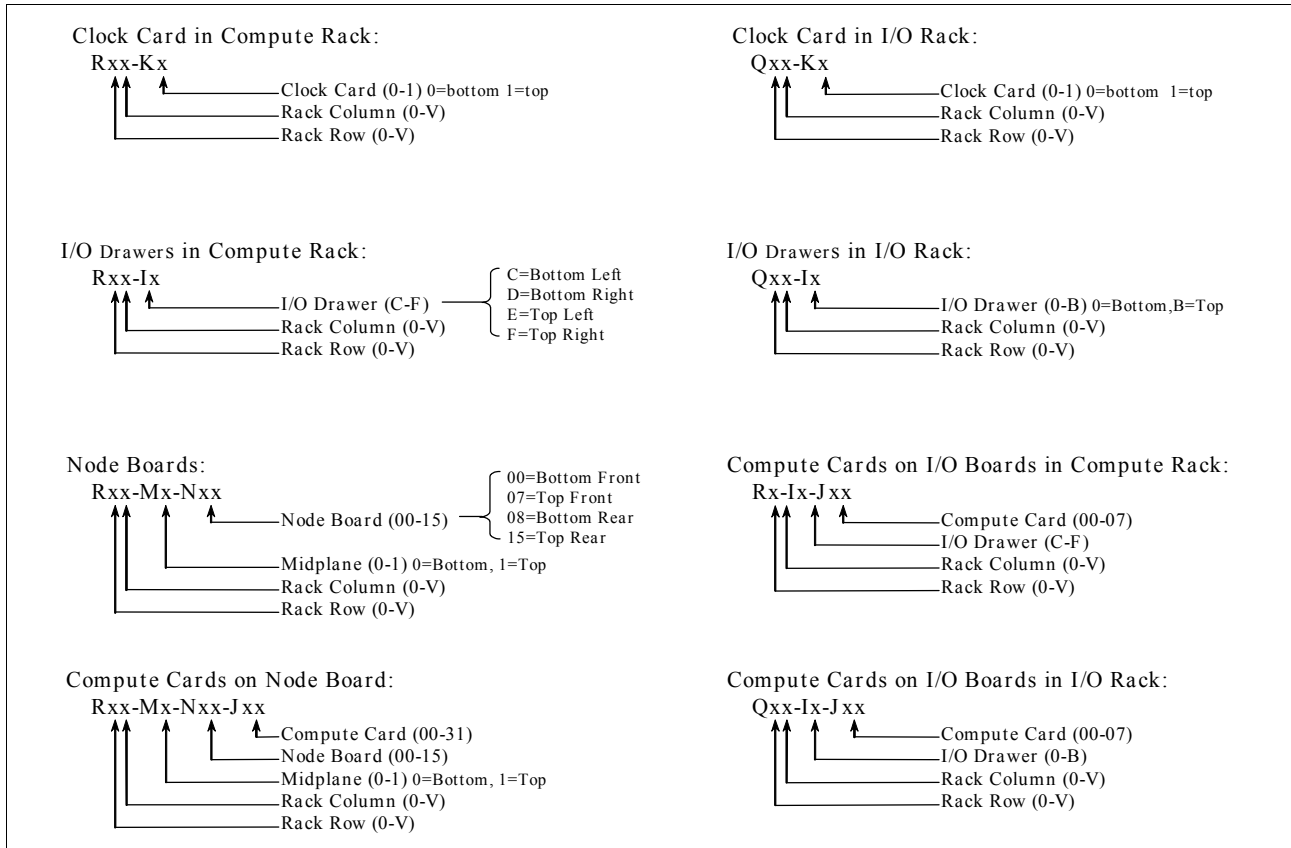


Figure A-2 Hardware naming convention for clock cards, I/O drawers, node boards and compute cards

Figure A-3 shows the Blue Gene/Q naming convention used when assigning locations for cores, link modules, and Direct Current Assemblies (DCA).

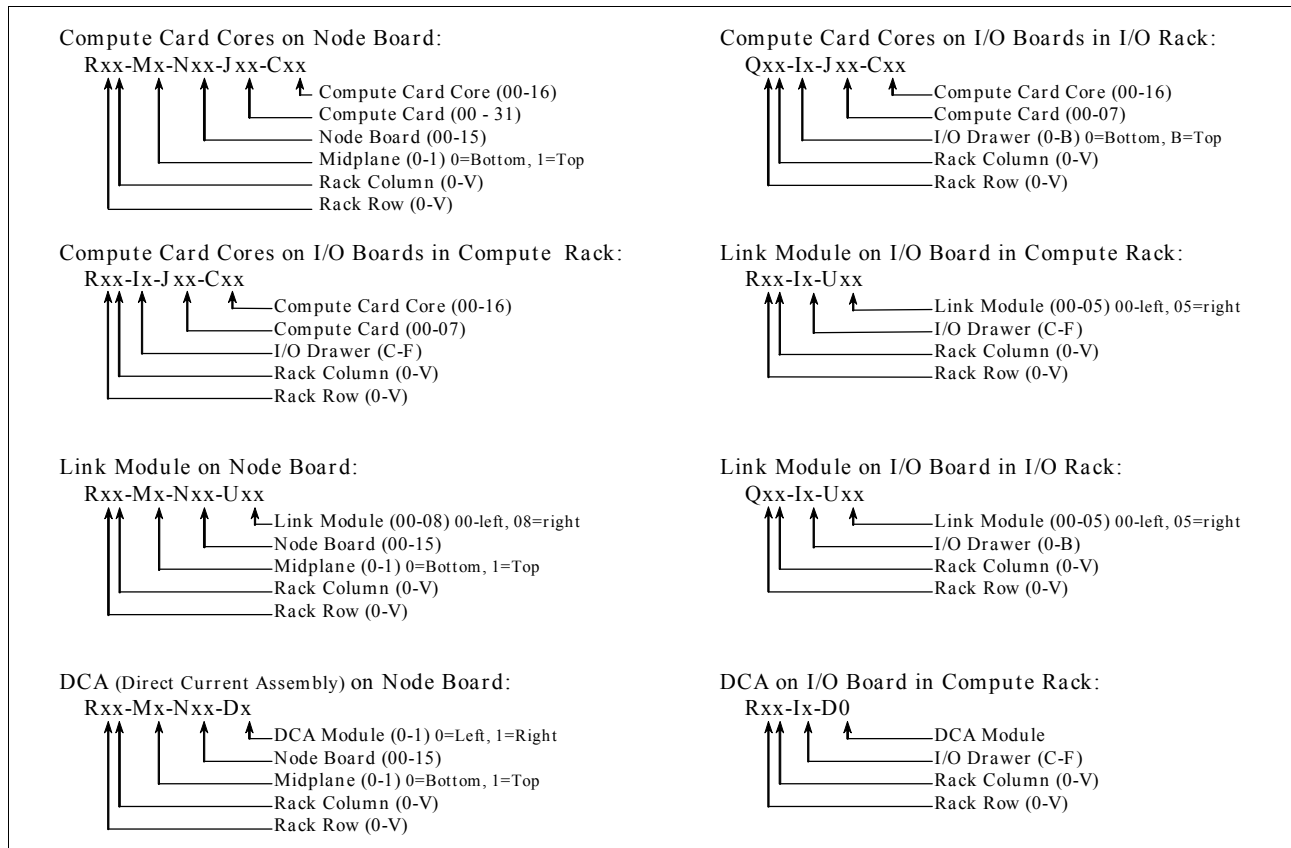


Figure A-3 Hardware naming convention for cores, link modules and DCA

Figure A-4 shows the Blue Gene/Q naming convention used when assigning locations for optical modules, fans, and DCA.

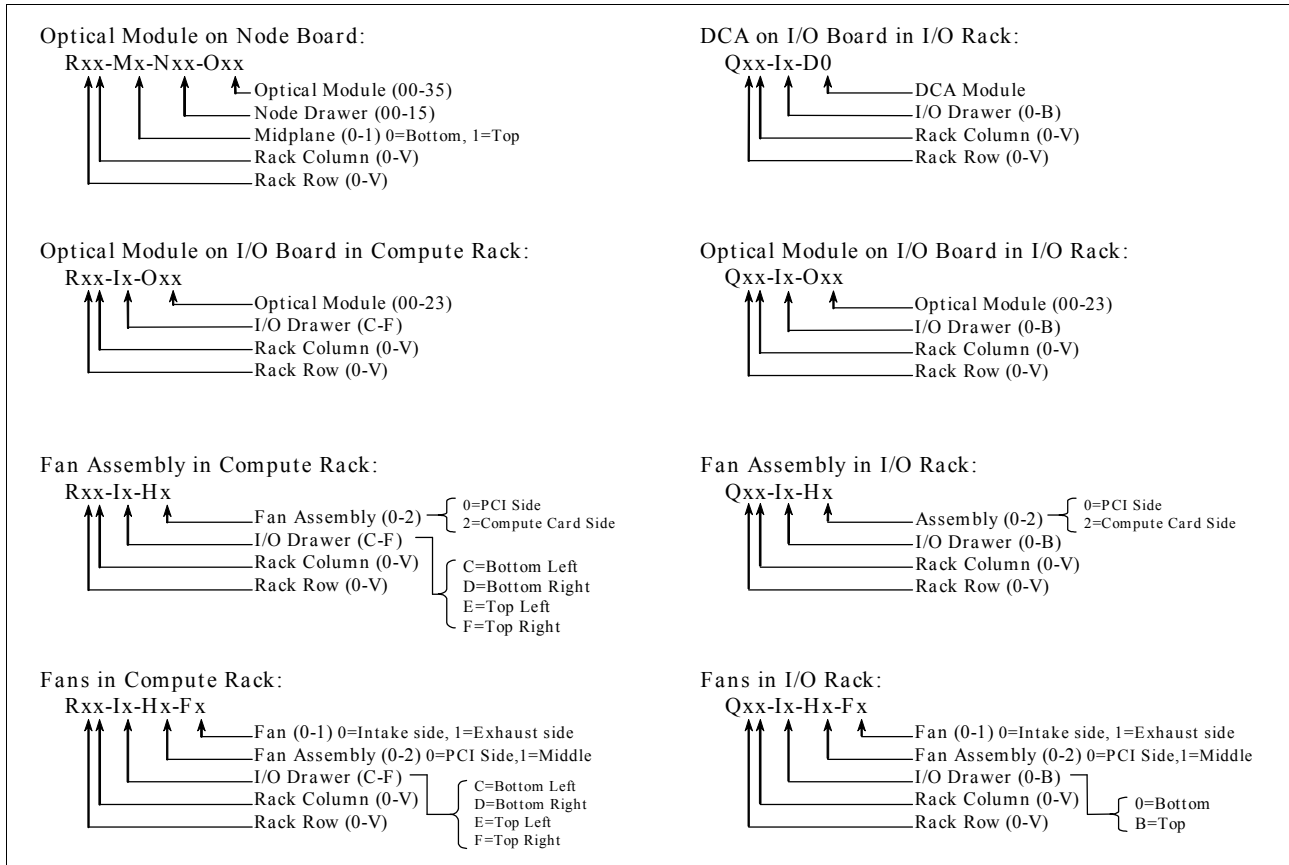


Figure A-4 Hardware naming convention for optical modules, fans and DCA

Figure A-5 shows the Blue Gene/Q naming convention used when assigning locations for PCI adapters and coolant monitors.

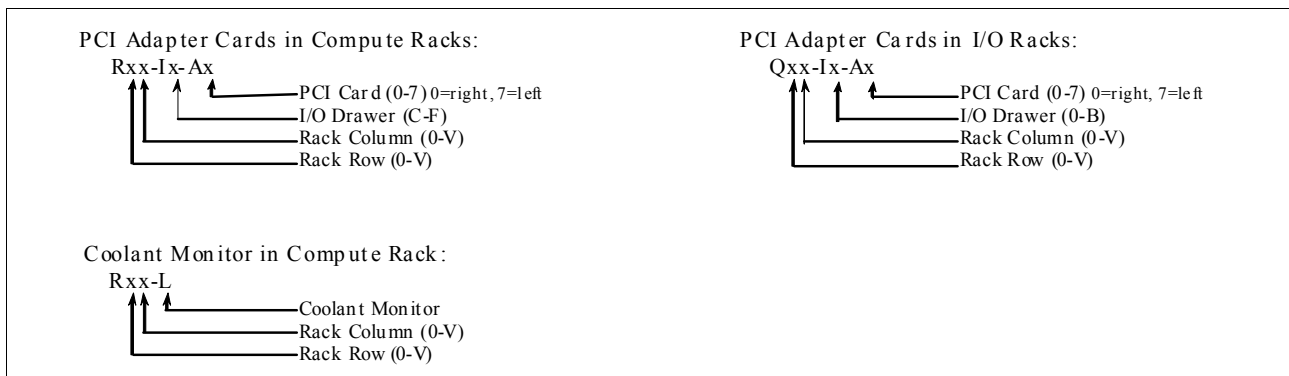


Figure A-5 Hardware naming convention for PCI adapters and coolant monitors

As an example, if you had an error in fan R23-IC-H1-F0 where would you go to look for it? The upper left corner of Figure A-1 on page 242 shows that the compute racks use the convention Rxx. Looking at the error message, the rack involved is R23. From the chart, that rack is the fourth rack in row two (remember that all numbering starts with 0). The bottom left corner of Figure A-4 indicates the bottom left I/O drawer of any rack is C. The chart shows that in the Fan Assemblies description, assembly 1 is in the middle. So, you are going to be checking for

an attention light (Amber LED) on the middle Fan Assembly on the intake side because that fan is causing the error message to surface.

Table A-1 contains several examples of hardware location names.

Table A-1 Examples of location names

| Card/board            | Element | Name                                          | Example        |
|-----------------------|---------|-----------------------------------------------|----------------|
| Compute on node board | Card    | J00 through J31                               | R23-M1-N02-J09 |
| Link on node board    | Module  | U00 through U08 (00 left most, 08 right most) | R32-M0-N04-U03 |
| Service               | Card    | S                                             | R00-M0-S       |

Figure A-6 shows the layout of a system with 64 compute racks and 16 I/O racks and how the compute racks and I/O rack locations are named.

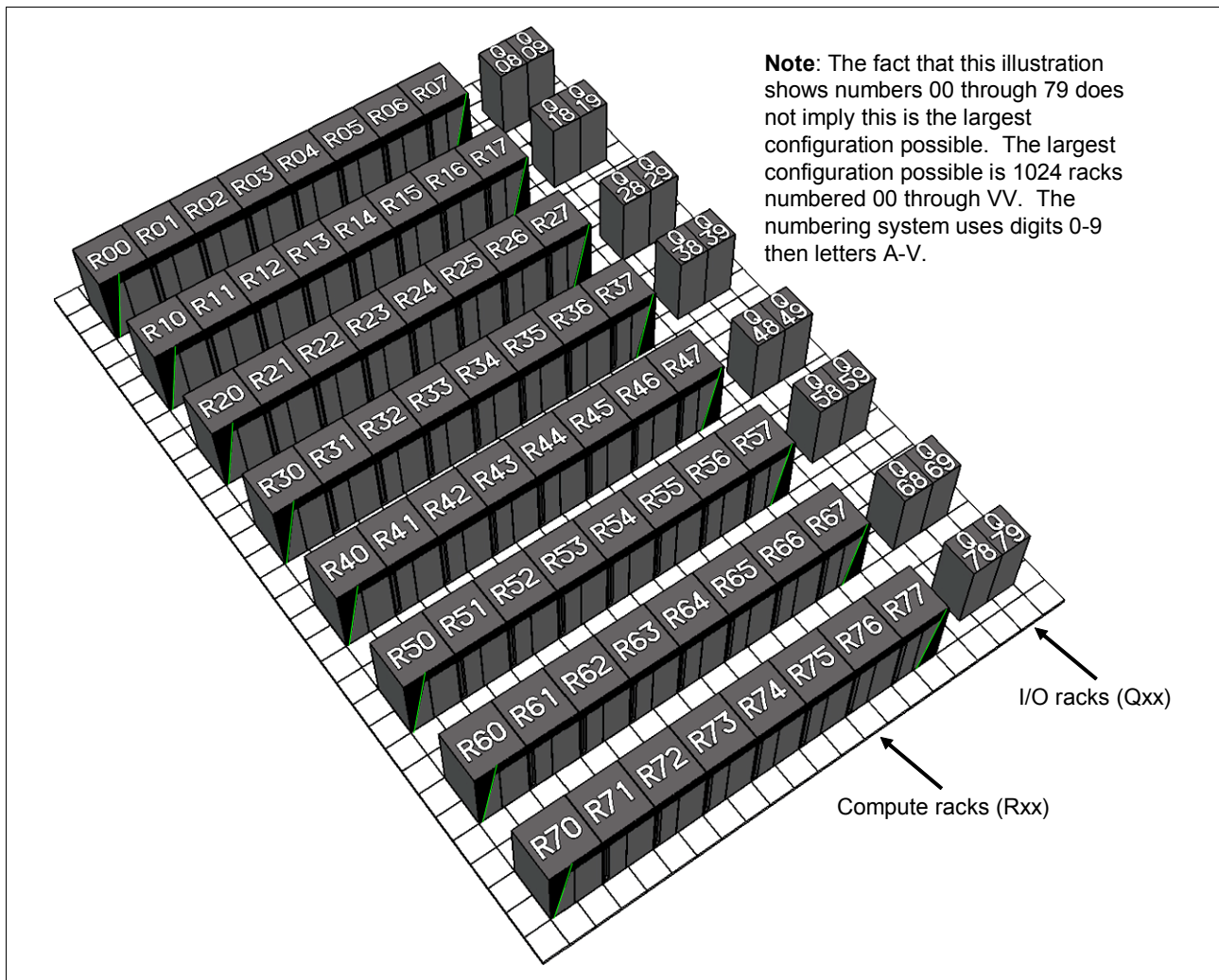


Figure A-6 Rack numbering



**B**

## Control System simulator

The Blue Gene/Q Control System simulator environment is typically used by job scheduler developers to test out system configurations and Application Programming Interfaces (APIs) without requiring actual Blue Gene/Q hardware to be present. Except for some restrictions on running jobs, the majority of *High-Level Control System* (HLCS) software behaves exactly like a real Blue Gene/Q production environment.

The simulator leverages the Blue Gene/Q database to simulate jobs running on essentially any supported machine configuration (for example a 16 rack system). It achieves this by simulating the I/O node software stack with four processes per I/O node. Even though it is possible to simulate large system configurations, there are a few limitations, such as the number of simulated jobs running at one time.

## Setting up

Perform the following steps to set up the simulator environment. Prior to this step, it is expected that a system acting as combination service node and front end node has been installed with the necessary software and the DB2 database has been set up but not populated with any data.

1. Create your own bg.properties file:

```
user@host ~> cp /bgsys/drivers/ppcfloor/utility/etc/bg.properties.ship.tpl
/pathtoyourcfg/bg.properties
user@host ~> export BG_PROPERTIES_FILE=/pathtoyourcfg/bg.properties
```

2. Edit your bg.properties file:

- Add **schema\_name = user** in the [database] section, where user is your user name. You must have authority to the DB2 database. See section 18.3, “Database security” on page 210 for instructions about setting up database authority.
- Pick a machine size to simulate by changing the **computeRackRows** and **computeRackColumns** values in the [database] section.
- You must add the **max\_user\_processes** key to the [runjob.server] section. By default, you must set the value to 256 (**max\_user\_processes = 256**). This configuration setting is designed to prevent overloading a system with simulated I/O processes forked from runjob\_server. There is no maximum value beyond what the Linux kernel enforces for non-privileged users. Be careful when changing this value.
- Change port numbers to be unique in the following sections:
  - [runjob.server] **command\_listen\_ports** and **mux\_listen\_ports**
  - [runjob.mux] **host** should match the port from [runjob.server] **mux\_listen\_ports**
  - [runjob.server.commands] **host** should match the port from [runjob.server] **command\_listen\_ports**
  - [runjob.mux] **local\_socket** must contain something specific to your uid, for example: **local\_socket = runjob\_mux.username**
- Add the following simulation settings in the [runjob.server] section:
 

```
job_sim = true
iosd_sim_path = /bgsys/drivers/ppcfloor/ramdisk/bin
iosd_sim_name = start_job_simulation
iosd_id_counter_name = /tmp/runjob_server_iosd_counter
iosd_id_counter_size = 2048
iosd_remove_log_files = true
```
- Modify the binary argument in the [master.binargs] section for mmcs\_server so it will not perform environmental monitoring. This is done by specifying the --no-poll-env argument. Environmental monitoring can also be turned off by setting poll\_envs = false in the [mmcs.envs] section of bg.properties. In addition, specify the location of the bg.properties file to use when starting the various servers by passing in the --properties argument:

```
[master.binargs]
mmcs_server = --properties /pathtoyourcfg/bg.properties --no-poll-envs
runjob_mux = --properties /pathtoyourcfg/bg.properties
bgmaster_server = --properties /pathtoyourcfg/bg.properties
bgws_server = --properties /pathtoyourcfg/bg.properties
realtime_server = --properties /pathtoyourcfg/bg.properties
runjob_server = --properties /pathtoyourcfg/bg.properties
mc_server = --properties /pathtoyourcfg/bg.properties
```

- Modify the mc\_server binary mapping in the [master.binmap] section by changing mc\_server so it maps to mc\_serversim instead of mc\_server\_64:

```
[master.binmap]
bgmaster_server=/bgsys/drivers/ppcfloor/hlcs/sbin/bgmaster_server
bgws_server=/bgsys/drivers/ppcfloor/bgws/sbin/bgws_server
realtime_server=/bgsys/drivers/ppcfloor/hlcs/sbin/bg_realtime_server
runjob_server=/bgsys/drivers/ppcfloor/hlcs/sbin/runjob_server
runjob_mux=/bgsys/drivers/ppcfloor/hlcs/sbin/runjob_mux
mc_server=/bgsys/drivers/ppcfloor/control/sbin/mc_serversim
mmcs_server=/bgsys/drivers/ppcfloor/hlcs/sbin/mmcs_server
subnet_mc=/bgsys/drivers/ppcfloor/control/sbin/SubnetMc_64
```

3. Create a database schema:

```
user@host ~> /bgsys/drivers/ppcfloor/db/schema/createBGQSchema
/dbhome/bgqsysdb/sqllib bgdb0
```

4. Populate your database schema:

This will use the **rackRows** and **rackColumns** values that you specified in your bg.properties file to pick a machine configuration.

```
user@host ~> source /dbhome/bgqsysdb/sqllib/db2profile
user@host ~> cd /bgsys/drivers/ppcfloor/db/schema
user@host schema> ./dbPopulate.pl --properties /pathtoyourcfg/bg.properties
```

## Starting and stopping servers

To simulate booting blocks and running jobs, four servers are required to be running in the simulator environment. They are mc\_serversim, mmcs\_server, runjob\_server, and runjob\_mux. Except for mc\_serversim, these are all standard servers in a production environment. Because the simulator has no actual Blue Gene/Q hardware that is being managed, the normal mc\_server\_64 binary is replaced with a special executable called mc\_serversim that supports the simulator setup. Swapping out mc\_server\_64 with mcserver\_sim was a step in setting up the simulator. In addition, the bg.properties file was updated so servers are started using the customized bg.properties configuration file.

After all of the bg.properties updates are completed, the required servers can be started using:

```
master_start bgmaster --properties /pathtoyourcfg/bg.properties
```

For more details about `master_start`, see section 11.2.1, “The `master_start` command” on page 131.

Servers can be stopped using:

```
master_stop bgmaster --properties /pathtoyourcfg/bg.properties
```

For more details about `master_stop`, see section 11.2.10, “The `master_stop` command” on page 136.

## Creating and booting I/O blocks and compute blocks

Before you can run a job in the simulator environment, you need both an I/O block and a compute block booted. Details about creating I/O blocks and compute blocks is in Chapter 3, “Compute and I/O blocks” on page 29.

After the I/O block and compute block are created, the next step is to boot them. Blocks can be booted using the `allocate` command from the `bg_console`. For this example, assume the I/O block name created is “IOBlock” and that the compute block name created is “R00-M0-N00”. Note that the I/O block must be booted before the compute block.

- ▶ Booting I/O block from `bg_console`:

```
mmcs$ allocate IOBlock
OK
mmcs$
```

- ▶ Booting compute block from `bg_console`:

```
mmcs$ allocate R00-M0-N00
OK
mmcs$
```

## Running jobs

Jobs can be submitted after the I/O block and compute block are successfully booted. Job submission is done using the standard `runjob` command. For complete details about the `runjob` command, see section 6.2, “The `runjob` command” on page 76.

In this example, the `/bin/date` executable is run on compute block R00-M0-N00.

```
user@host ~> /bgsys/drivers/ppcfloor/hlcs/bin/runjob --block R00-M0-N00 --properties
/pathtoyourcfg/bg.properties : /bin/date
```

## Limitations with `runjob`

There are a few limitations with job simulation. Some are inherent in the overall idea of simulating such a large supercomputer, and others are intentionally omitted:

- ▶ No support is provided for `stdin`.
- ▶ Alternative mappings through the `runjob --mapping` argument are not supported.

- ▶ The tool arguments (for example, `--start-tool`, `--tool-args`, and so on) of `runjob` are not supported.

When simulating large configurations, you might see the following entry in the `runjob_server` output before it aborts:

```
2010-05-03 02:57:06.668 (ERROR) [0x4000c81f1b0]
R00-IC-J00:ibm.runjob.server.sim.Iosd: could not fork 11 (Resource temporarily
unavailable)
```

This is due to a safety mechanism designed to prevent overloading the system. The `runjob_server` unconditionally sets the maximum number of user processes with the following value during job simulation when it starts:

```
setrlimit(RLIMIT_NPROC, value)
```

You can modify this value using the `max_user_processes` key in the `[runjob.server]` section of your `bg.properties` file. The default value is 256. If you run a `vnc`, you probably want to increase this value. Be careful when increasing this to a large value because it can lead to overloading your system.

## Simulator cleanup

Job simulation normally cleans up processes but it is a good practice to check for any stray processes just in case cleanup was not done. The following commands clean up any stray processes:

```
user@host ~> killall -KILL iosd
user@host ~> killall -KILL jobctld
user@host ~> killall -KILL stdiod
```

Temporary `iosd` directories and log files must also be removed by running the following command:

```
user@host ~> rm -rf /tmp/cios*
```



# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

## IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 254. Note that some of the documents referenced here might be available in softcopy only.

- ▶ *IBM System Blue Gene Solution: Blue Gene/Q Application Development Manual*, SG24-7948-00
- ▶ *IBM System Blue Gene Solution: Blue Gene/Q Code Development and Tools Interface*, REDP-4659
- ▶ *IBM System Blue Gene Solution: Blue Gene/Q Hardware Installation and Maintenance Guide*, SG24-7974-00
- ▶ *IBM System Blue Gene Solution: Blue Gene/Q Hardware Overview and Planning*, SG24-7872-00
- ▶ *IBM System Blue Gene Solution: Blue Gene/Q Safety Considerations*, REDP-4656
- ▶ *IBM System Blue Gene Solution: Blue Gene/Q Service Node Failover using Linux High Availability*, REDP-4657

## Other publications

This publication is also relevant as a further information source:

- ▶ *General Parallel File System HOWTO for the IBM System Blue Gene/Q Solution*, SC23-6939-00

## Online resources

These websites are also relevant as further information sources:

- ▶ Logging Services: Short introduction to Apache log4cxx  
<http://logging.apache.org/log4cxx/>
- ▶ pwauth: A Unix Authentication Tool  
<http://code.google.com/p/pwauth>
- ▶ SourceForge: Building pyodbc  
[http://sourceforge.net/apps/mediawiki/pyteal/index.php?title=Building\\_pyodbc](http://sourceforge.net/apps/mediawiki/pyteal/index.php?title=Building_pyodbc)
- ▶ SourceForge: HowTo - Event Analyzer  
[http://sourceforge.net/apps/mediawiki/pyteal/index.php?title=HowTo\\_-\\_Event\\_Analyzer](http://sourceforge.net/apps/mediawiki/pyteal/index.php?title=HowTo_-_Event_Analyzer)
- ▶ SourceForge: Toolkit for Event Analysis and Logging  
<http://sourceforge.net/apps/mediawiki/pyteal/index.php>

- ▶ Updating Firmware and Software for IBM Network Adapter Cards & Switch Systems  
[http://www.mellanox.com/content/pages.php?pg=firmware\\_table\\_IBM](http://www.mellanox.com/content/pages.php?pg=firmware_table_IBM)
- ▶ YACI: Yet Another Cluster Installer  
<http://www.yaci.org>

## How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and additional materials, as well as order hardcopy Redbooks publications, at this website:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)



# Abbreviations and acronyms

|                |                                               |                 |                                        |
|----------------|-----------------------------------------------|-----------------|----------------------------------------|
| <b>API</b>     | Application programming interfaces            | <b>SYSIOD</b>   | System I/O daemon                      |
| <b>BGWS</b>    | Blue Gene Web Services                        | <b>TCP</b>      | Transmission control protocol          |
| <b>BIST</b>    | Built-in self-test                            | <b>TEAL</b>     | Toolkit for Event Analysis and Logging |
| <b>BPE</b>     | Bulk power enclosure                          | <b>TOOLCTLD</b> | Tool control daemon                    |
| <b>BPM</b>     | Bulk power module                             | <b>UDP</b>      | User datagram protocol                 |
| <b>CA</b>      | Certificate authority                         | <b>YACI</b>     | Yet Another Cluster Installer          |
| <b>CIOS</b>    | Common I/O services                           |                 |                                        |
| <b>CN</b>      | Common name                                   |                 |                                        |
| <b>CNK</b>     | Compute Node Kernel                           |                 |                                        |
| <b>DCA</b>     | Direct current assembly                       |                 |                                        |
| <b>DCR</b>     | Device control registers                      |                 |                                        |
| <b>ELA</b>     | Event log analysis                            |                 |                                        |
| <b>FEN</b>     | Front end nodes                               |                 |                                        |
| <b>FPGA</b>    | Field programmable gate arrays                |                 |                                        |
| <b>GDB</b>     | GNU project debugger                          |                 |                                        |
| <b>GPR</b>     | General purpose registers                     |                 |                                        |
| <b>GUI</b>     | Graphical user interface                      |                 |                                        |
| <b>HLCS</b>    | High-Level Control System                     |                 |                                        |
| <b>HPC</b>     | High performance computing                    |                 |                                        |
| <b>IAR</b>     | Instruction address registers                 |                 |                                        |
| <b>IOSD</b>    | I/O services daemon                           |                 |                                        |
| <b>IP</b>      | Internet protocol                             |                 |                                        |
| <b>IPC</b>     | Inter-process communication                   |                 |                                        |
| <b>JOBCTLD</b> | Job control daemon                            |                 |                                        |
| <b>LAN</b>     | Local area network                            |                 |                                        |
| <b>LLCS</b>    | Low-Level Control System                      |                 |                                        |
| <b>MC</b>      | Machine controller                            |                 |                                        |
| <b>MDC</b>     | Mapped diagnostic context                     |                 |                                        |
| <b>MMCS</b>    | Midplane Management Control System            |                 |                                        |
| <b>NFS</b>     | Network file system                           |                 |                                        |
| <b>PID</b>     | Process ID                                    |                 |                                        |
| <b>QDR</b>     | Quad data rate                                |                 |                                        |
| <b>RAS</b>     | Reliability, availability, and serviceability |                 |                                        |
| <b>SN</b>      | Service node                                  |                 |                                        |
| <b>SPR</b>     | Special purpose registers                     |                 |                                        |
| <b>SSL</b>     | Secure sockets layer                          |                 |                                        |
| <b>SSN</b>     | Subnet service nodes                          |                 |                                        |
| <b>SSR</b>     | Systems service representative                |                 |                                        |
| <b>STDIOD</b>  | Standard I/O daemon                           |                 |                                        |



# Index

## A

alerts 5  
 allocate 38, 68–70, 147, 151, 221, 250  
 allocate\_block 70, 151, 221  
 API 186–187, 204  
 application programming interfaces 247  
 attention 37  
 authentication 187, 253

## B

BGQMaster  
   mcServer 160  
 BGWS 68, 70, 130, 132, 139, 223–224, 249  
 block 3, 38, 70, 215  
   creating in mmcs 35  
 blocks 5  
 Blue Gene Navigator 7, 145, 212, 214  
 Blue Gene Web Services 68, 130  
 boot\_block 70  
 BPE 158  
 BPM 155  
 buffer 137, 144  
 bulk power module 155–156, 158

## C

CA 186, 205–209  
 certificate authority 205–207, 210  
 CIOS 38, 221, 223, 229, 251  
 clients 70, 136, 148, 160, 185–188, 205, 207, 220  
 CN 149, 206–209  
 CNK 3  
 command 135, 137–139, 142, 200  
 commands 30, 130, 144, 178  
 common I/O services 221, 223, 229  
 common name 206, 208  
 compute card 241  
 Compute Node Kernel 3, 151  
 configuration 2, 131–141, 143–144, 148–150, 160–162, 186–190, 201, 204, 207, 210–212, 219–225, 227–228, 230, 247–249  
 configuring 224  
 connect 70–71, 130–131, 133–139, 143–144, 150, 160, 162, 188–189, 210, 212, 214, 216, 228  
 console 5, 30, 38, 67–69, 72–74, 131, 139, 147–148, 150, 153, 156, 160, 204–205, 221, 250  
 Coreprocessor tool 231–232  
 creating 190, 212, 250

## D

dbPopulate 212, 228, 249  
 DCA 156–157, 241, 244–245  
 delete 38, 70, 204, 210, 213–214

deselect\_block 70  
 diagnostics xiii, 5, 7, 160, 205, 207, 262  
 direct current assembly 241  
 disconnect 70  
 dynamic 30, 215, 230

## E

ending 140  
 environment variable 52, 72, 131–139, 150, 161–162, 187–189, 213  
 environmental monitor 152, 156  
 errors 70, 137–138, 144, 216  
 event analyzer 253

## F

free 38, 70, 149, 151  
 free\_block 70  
 front end nodes 1, 4

## G

GDB 4  
 genblock 35  
 General Parallel File System 4, 253  
 GNU project debugger 4  
 GPFS 4

## H

High-Level Control System 147, 247  
 HLCS 131–138, 141, 147, 149, 247, 249–250

## I

I/O node 3–4, 30, 37–38, 154, 224, 229, 247  
 internet protocol 4  
 IOSD 248, 251  
 IP xiii, 4, 130–139, 142–144, 149–150, 161, 186, 188, 200, 205, 228

## J

JOBCTLD 251

## K

kill\_job 72

## L

LAN 2  
 list 38, 69–74, 131, 135–136, 140–143, 145, 149–150, 152, 156, 186, 188, 200–201, 204–205, 215–216, 221–222, 228  
 list\_blocks 71, 73–74  
 list\_jobs 69–70

list\_users 71, 73–74  
 LLCS 68, 154, 159–160  
 local area network 2  
 locate 71–72  
 log file 52, 186, 219–221  
 logging 4, 130, 139, 161–162, 186–190, 219–220, 222–223, 227  
 logs 104, 139, 149–150, 185–186, 189, 219–222, 224–225  
 Low-Level Control System 68, 154, 159–160

## M

machine controller 68, 130, 159–160, 223  
 mailbox 151  
 mapped diagnostic context 220  
 MC 68, 70, 129–130, 132, 141–142, 147–148, 150–151, 159–162, 200, 223–224, 249  
 mcServer 70, 148, 160, 223, 249  
 MDC 220  
 mesh 31  
 messages 94, 137, 145, 147, 153  
 Midplane Management Control System 68, 130, 147  
 MMCS 30, 35, 38, 68–73, 130, 132, 147–154, 160, 220–221, 223–224, 230, 248–250  
 mmcs 30, 35, 38, 68–73, 130, 132, 147–154, 160, 220–221, 223–224, 230, 248–250  
     mcServer 148  
 MMCS commands 148

## N

Navigator 7, 30, 35, 38, 145, 212, 214  
     diagnostics 123  
 navigator 5  
 network 3–4, 143–144, 203, 205  
 network file system 4  
 NFS 4

## P

pass through 31, 34, 36  
 performance 152, 154, 212–215, 229  
 PID 132–134  
 pipes command input 72  
 process ID 130–131, 133–135

## Q

quit 71

## R

ramdisk 4, 248  
 RAS 3, 38, 52, 94, 137, 144–145, 147–148, 151–158, 160–161, 186, 213–214, 223, 229  
 read 70–71, 188–189, 221–222  
 Real-time 130, 187, 190  
 real-time 68, 185–190, 207, 223  
 reboot\_nodes 71  
 Redbooks website 254  
     Contact us xiv

redirect 71, 151  
 reliability, availability, and serviceability 3, 147

## S

secure sockets layer 187, 205  
 select\_block 71  
 Service Action  
     logs 117  
 service actions xiii, 5, 7, 160, 262  
 service node 1–5, 230  
 setup 248–249  
 severity 94, 220  
 shutdown 149, 151, 161  
 sleep 71  
 small blocks 30, 34  
 SN 1–2  
 socket 151, 248  
 sql 71, 189, 212, 215–216  
 SSL 187, 205  
 SSN 2  
 starting 139, 214, 248–249  
 startup 220, 230  
 static 30  
 static block 30  
 status 38, 69–70, 72, 131–133, 136–137, 139, 151, 156–158, 186–188, 212  
 STDIOD 251  
 stopping 139, 150, 249  
 subnet service nodes 2  
 system logs 219–220, 224

## T

TCP xiii, 4, 131–139, 149–150, 161, 205  
 TEAL 104, 130, 132, 224  
 Toolkit for Event Analysis and Logging 130, 253  
 transaction logs 185–186, 189  
 transmission control protocol 4

## U

UDP 4  
 user datagram protocol 4  
 username 72, 248

## V

version 72–73, 162, 215

## W

wait\_boot 72, 149  
 write 139  
 write\_con 72

## Y

YACI 254  
 Yet Another Cluster Installer 254

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 September 17, 2012 11:11 am

**7869spine.fm 259**



## IBM System Blue Gene Solution: Blue Gene/Q System Administration

(0.2"spine)  
0.17"<->0.473"  
90<->249 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 September 17, 2012 11:11 am

**7869spine.fm 260**





# IBM System Blue Gene Solution

## Blue Gene/Q System Administration



**Perform Blue Gene/Q system administration**

**Configure and use Blue Gene Navigator**

**Run diagnostics and perform service actions**

This IBM Redbooks publication is one in a series of books that are written specifically for the IBM System Blue Gene supercomputer, Blue Gene/Q, which is the third generation of massively parallel supercomputers from IBM in the Blue Gene series. This book provides an overview of the system administration environment for Blue Gene/Q. It is intended to help administrators understand the tools that are available to maintain this system.

This book details Blue Gene Navigator, which has grown to be a full featured web-based system administration tool on Blue Gene/Q. 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 BGmaster and the Control System processes that it monitors.

This book is intended for Blue Gene/Q system administrators. It helps them use the tools that are available to maintain the Blue Gene/Q system.

### **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](http://ibm.com/redbooks)