

Deployment Guide for Advanced Monitoring of a Blue Gene Environment

See the IBM Blue Gene solution
overview

Learn to install and configure a
Blue Gene environment

Take advantage of code
scripts for plug-ins



James Doyle
Matthew Holt
Cindy Mestad
Steve Westerbeck



International Technical Support Organization

**Deployment Guide for Advanced Monitoring of a Blue
Gene Environment**

January 2008

Archived

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

First Edition (January 2008)

This edition applies to the IBM® Blue Gene® Solution.

© Copyright International Business Machines Corporation 2008. 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	v
Trademarks	vi
Preface	vii
The team that wrote this paper	vii
Become a published author	viii
Comments welcome	viii
Chapter 1. The monitoring solution overview	1
1.1 Introduction to system monitoring	2
1.2 IBM Blue Gene solution	4
Chapter 2. Nagios, Fruity, and NRPE installation	5
2.1 Prerequisites	6
2.2 Nagios	6
2.3 Fruity	7
2.4 NRPE	7
Chapter 3. Apache WebServer, MYSQL, Nagios, Fruity, and NRPE configuration	9
3.1 Configuring Apache to work with Nagios	10
3.2 MySQL	10
3.3 Installing Fruity	11
3.4 Configuring Nagios using the Web interface	11
3.5 Configuring NRPE	13
Chapter 4. Nagios implementation and its plug-ins	15
4.1 Nagios	16
4.1.1 Commands	16
4.1.2 Time periods	16
4.1.3 Contacts	17
4.1.4 Templates	18
4.1.5 Hosts	19
4.1.6 Services	19
4.2 Plug-ins	20
4.2.1 Development	20
4.2.2 Resources	20
4.2.3 Included plug-ins	22

4.3 Sample Nagios screen captures	23
Part 1. Appendixes	29
Appendix A. Blue Gene/L plug-ins	31
A.1 Check BG/L jobs	32
A.2 Check BG/L link card	37
A.3 Check BG/L midplane	40
A.4 Check BG/L node card	44
A.5 Check BG/L node	47
A.6 Check BG/L processor card	51
A.7 Check BG/L service action	54
A.8 Check BG/L service card	58
Appendix B. Blue Gene/P plug-ins	63
B.1 Check BG/P jobs	64
B.2 Check BG/P link card	69
B.3 Check BG/P midplane	72
B.4 Check BG/P node	76
B.5 Check BG/P node card	79
B.6 Check BG/P service action	83
B.7 Check BG/P service card	86
Appendix C. DB2 plug-ins	91
C.1 DB2 health check	92
Appendix D. Other plug-ins	97
D.1 Check daemon	98
D.2 Who users	100
Appendix E. Plug-in resources	103
E.1 Event handlers	104
E.1.1 Event handler template	104
E.1.2 Check Blue Gene (BG) jobs event handler	105
E.1.3 Other Check BG event handlers	107
E.1.4 Check DB2 Health Event Handler	116
E.2 Build HTML	119
E.3 Check BG jobs reset	120
E.4 Check DB2 health reset	121
Appendix F. NRPE scripts	125
F.1 Install NRPE	126
F.2 Reinstall NRPE	128
F.3 Remove NRPE	130
F.4 NRPE Manager	132
Appendix G. Additional material	135
Locating the Web material	135
Using the Web material	135
System requirements for downloading the Web material	135
How to use the Web material	136

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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


This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

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

Redbooks (logo) ®
iSeries®
AIX®
AS/400®

Blue Gene/L™
Blue Gene/P™
Blue Gene®
DB2®

DS4000™
IBM®
Redbooks®

The following terms are trademarks of other companies:

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

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

Preface

The larger a data center grows, the more monitoring it requires. But limited staff, mixed hardware, multiple software configurations, and complex network topologies all combine to increase the complexity of monitoring. In addition, high-dollar monitoring solutions require cost justification, additional training, higher software budgets, and pricey maintenance contracts, and they can add their own idiosyncrasies to a complex environment. Adding more people to provide 24/7 operational coverage is not cost-effective, except in the largest computing environments.

How can we make the most effective use of limited technical staff, alert the proper resource to occasional but critical system errors or potential problems in the making, and enhance troubleshooting? In addition, every data center is faced with updated architectural requirements and unanticipated configuration changes. Constant requests for new hardware and software requirements must be weighed against limited budgets and the need for uptime.

Nagios offers a flexible solution: a widely supported and extensible monitoring application. And it is free. Teaming Nagios up with the renowned IBM® Blue Gene® system creates an attractive combination.

The team that wrote this paper

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

James Doyle was an IBM summer intern and is a senior at the University of Nebraska in Lincoln. His major is Computer Science. His hometown is Austin, MN.

Matthew Holt was an IBM summer intern and is a senior at Wake Forest University, where he is majoring in Computer Science. His hometown is Burlington, NC.

Cindy Mestad is the Senior Project Manager of the Blue Gene Center in Rochester, MN, with a mission to provide remote access for IBM employees and clients to the Blue Gene Supercomputer for presales benchmarking, ISV application enablement, and proof of concept testing. Cindy joined IBM in 1980 and has worked in AS/400® Early Programs, AS/400 Systems Performance, and the iSeries® Benchmark Center. She became PMI-certified in Project Management in 2003.

Steve Westerbeck is an Advisory Software Engineer and System Administrator at the IBM Rochester Blue Gene Center. He has been involved in the IT industry for over 20 years, including projects involving the i5/OS and OS/400 operating systems, AIX®, and Linux® system integration. He has also worked on multiplatform database development. He was an IT Manager for two city governments.

Also, thanks to the following person for contributing to this project:

Mike Ebbers has been with IBM for 34 years and writes for the ITSO on a variety of subjects. He is in the IBM International Technical Support Organization in Poughkeepsie.

Become a published author

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

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

Find out more about the residency program, browse the residency index, and apply online at:
ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- Send your comments in an e-mail to:

redbooks@us.ibm.com

- Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400



The monitoring solution overview

This paper shows you how to implement a low-cost system monitoring application that has the capability of working with both simple and complex data center environments. System Administrators often have the task of maximizing system uptime within mission critical environments. Yet they either are not provided with the human resources to do so, or they lack the funding to implement enterprise solutions. The solution described in this paper solves both of these constraints. Not only can this solution increase the efficiency of System Administrators and allow them to direct their time elsewhere, but it also is free of charge.

1.1 Introduction to system monitoring

The solution in this book is to implement and extend an Open Source host and service monitoring tool named Nagios. Nagios has been developed for several years and is very widely used. Due to its popularity as an Open Source application, it has large community support, which makes it easy for you to receive help through documentation, forums, mailing lists, and Internet relay chats (IRCs). Also, numerous plug-ins exist that offer extensive monitoring of many services that you otherwise must implement yourself.

Figure 1-1 shows how you can monitor a simple environment of workstations.

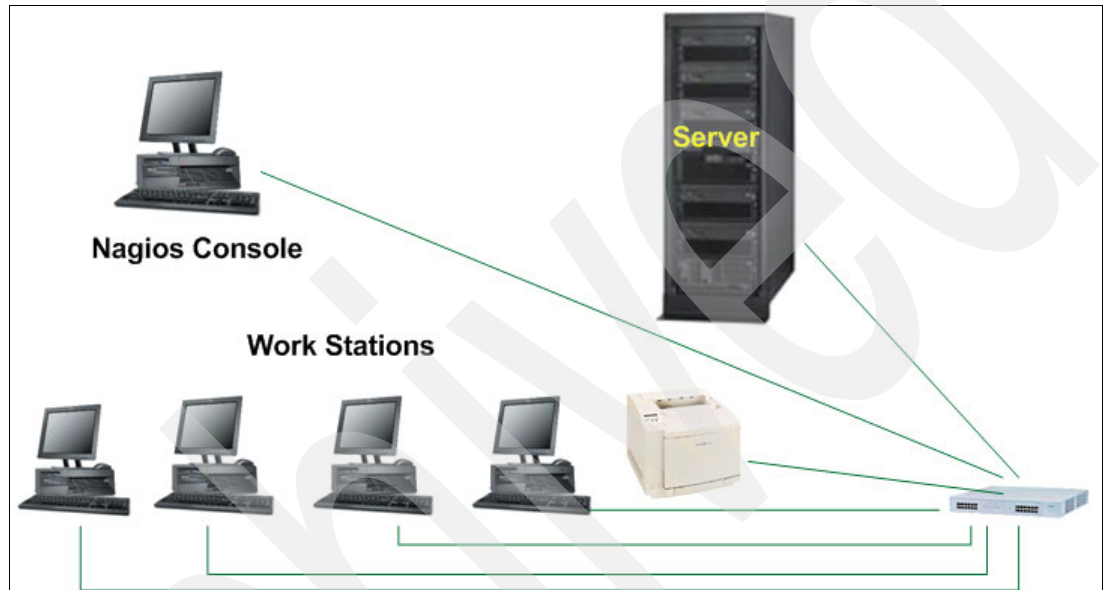


Figure 1-1 Monitoring a simple environment

Monitoring a highly complex data center environment might look like Figure 1-2 on page 3.

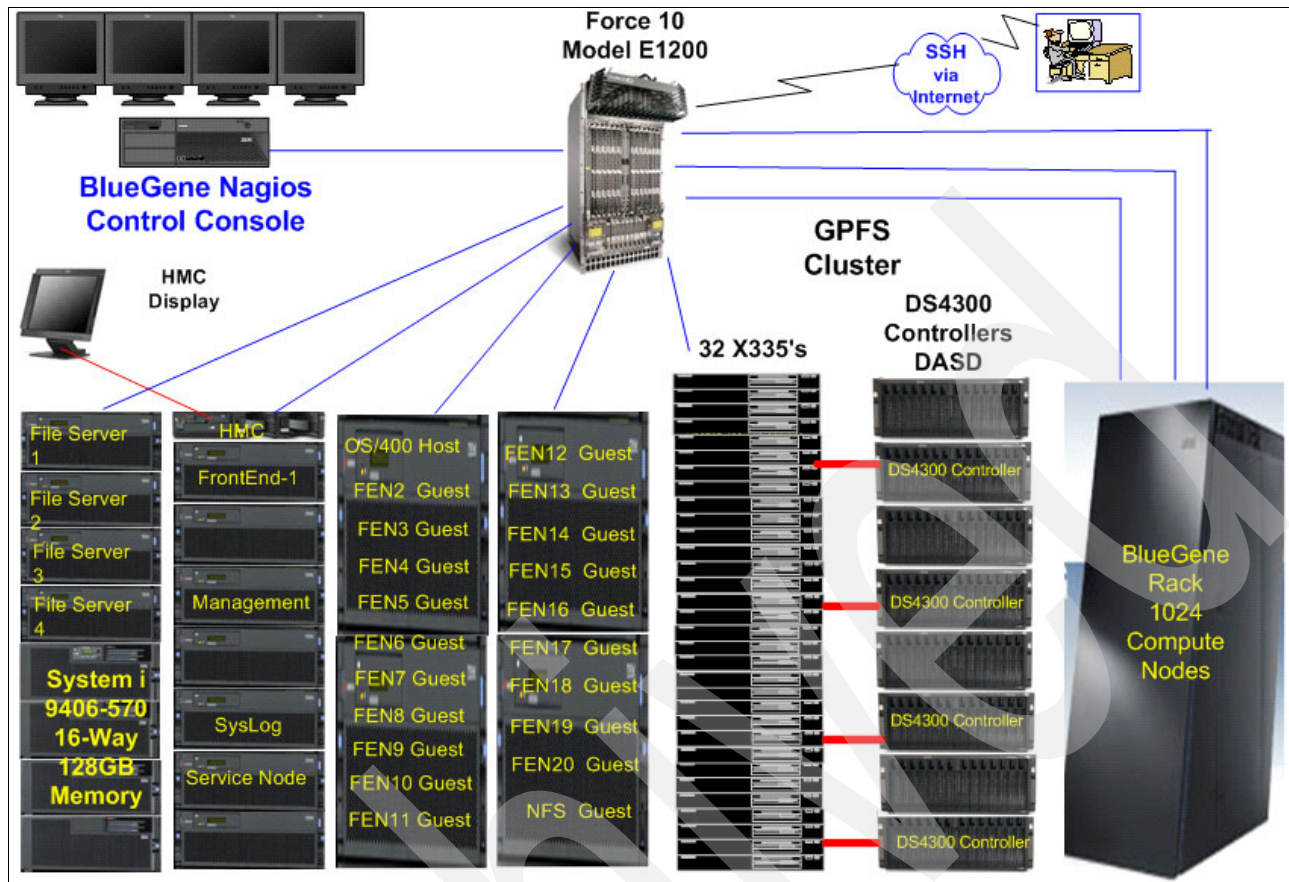


Figure 1-2 Monitoring a complex environment

One concern about Nagios is that its configuration is not trivial. The entire configuration is left to text files read by Nagios. Configuring the hosts and services of Nagios is definitely doable; however, it could be quicker. Thanks to the supporting community, several graphical user interface (GUI) configuration tools have been developed for Nagios. A section of this paper details the configuration of Nagios via a tool named Fruity.

Nagios itself works by monitoring all specified hosts via one of several methods available (SNMP, NRPE, NSCA, and more) and then reporting any issues back to the master host. The master host then displays the issues via a Web interface and has the ability to notify the administrators of the situation via e-mail or a text message. The way that Nagios actually monitors the hosts (or devices) is through one of several methods. This paper primarily focuses on using one method, the NRPE daemon. The NRPE daemon is installed on all hosts that are to be monitored. After a host is configured within Nagios, Nagios communicates with the host's NRPE daemon via SSL, which in turn checks the specified service and returns the result to Nagios. When using NRPE, a service is typically checked by calling an *executable* (or script) that actually performs the check. Thus, most things that fall within the realm of systems administration can be monitored via a service. This paper provides several Perl scripts that you can use as service checks.

Throughout this paper, we will describe the installation and configuration process of Nagios in detail. In this paper, we configure Nagios to monitor an example environment that contains a Blue Gene supercomputer. We also demonstrate how to monitor Linux machines, along with their respective services, and even a database. We provide scripts and instruction that will not only illustrate the monitoring of a Blue Gene supercomputer but will also enable you to monitor a wide variety of environments.

1.2 IBM Blue Gene solution

This paper addresses a computer monitoring solution for a Blue Gene supercomputer environment.

Today, Blue Gene ranks as the number one fastest supercomputer on the TOP500 list and holds 19 of the top 100 positions. When first chartered over five years ago, the goal of the Blue Gene project was to develop a massively parallel computer applied to the study of biomolecular phenomena, such as protein folding. The purpose of this effort was to advance the understanding of the mechanisms behind protein folding via large-scale simulation and to explore novel ideas in massively parallel machine architecture and software. The level of performance provided by Blue Gene can enable a tremendous increase in the scale of simulations beyond what is possible with other supercomputers. Successful simulation studies of protein folding on this scale are expected to advance the techniques, models, and algorithms used in biomolecular simulation.

Blue Gene clients note that time-to-solution for many applications has been reduced by an order of magnitude. Scientists can make fewer runs more often, which allows them to explore alternative models and approaches to problems. Blue Gene is making a demonstrable change in the way science is conducted.

Blue Gene delivers more than ultrascale performance. Because of unique design points that allow dense packaging of processors, memory, and interconnections, Blue Gene offers leadership efficiency in the areas of power and floor space consumption. Available in configurations ranging from one to 64 racks, Blue Gene is the innovative new solution from IBM that delivers an ultrascale solution without sacrificing efficiency.

For more information, go to:

<http://www-03.ibm.com/servers/deepcomputing/bluegene1.html>



Nagios, Fruity, and NRPE installation

If you have any difficulties with the instructions provided for installing any of the following packages, consult the official documentation for that package. It will give more in-depth instructions and probably includes troubleshooting tips.

2.1 Prerequisites

You must install these packages before proceeding:

- ▶ Apache HTTP Server 2.0+
- ▶ MySQL 5.0+

To start, create a user that will own and run most of the packages listed here. Make sure that this user has full **sudo** rights and can access any command via **sudo**. If this compromises security protocol for your environment, you can limit the user to having access to the **ssh**, **scp**, **chmod**, **chown**, **su**, and **ping** commands. A good example of the user/group is `nagios.nagcmd`.

2.2 Nagios

Nagios is the heart of this installation. It is the software that actually monitors your environment. See illustrations of its capabilities in 4.3, “Sample Nagios screen captures” on page 23.

You can download the latest version of Nagios from:

<http://www.nagios.org/download>

To install Nagios:

1. Move the tarball to the Nagios Master Server.
2. Extract the tarball:

```
tar xzf nagios-version.tar.gz
```
3. Create the directory into which you want to install.
4. Change its owner to the user/group combination created earlier (referred to from here on as the “nagios user” and “nagios group”):

```
chown nagios_user.nagios_group DIRECTORY
```
5. Identify the user that the Apache daemon runs as. It is listed in Apache’s `httpd.conf` file:
 - a. Add the Apache daemon user to the nagios group.
6. Move into the extracted tarball directory.
7. Issue this command using the following explanations:

```
./configure -prefix=prefix -with-cgiurl=cgiurl -with-htmurl=htmurl  
-with-nagios-user=someuser -with-nagios-group=somegroup  
-with-command-group=cmdgroup
```

 - a. Replace *prefix* with the installation directory that you created in the earlier step (the default is `/usr/local/nagios`).
 - b. Replace *cgiURL* with the actual URL that you will use to access the CGIs (the default is `/nagios/cgi-bin`). Do *not* append a slash at the end of the URL.
 - c. Replace *htmURL* with the actual URL that you will use to access the HTML for the main interface and documentation (the default is `/nagios/`).
 - d. Replace *someuser* with the name of the nagios user that you created earlier.
 - e. Replace *somegroup* with the name of the nagios group that you created earlier.
 - f. Replace *cmdgroup* with the same group name that you used previously.

8. Compile binaries:
`make all`
9. Install binaries in the specified directory:
`make install`
10. Install init scripts:
`make install init`
11. Download the latest version of the Official Nagios Plug-ins at:
<http://www.nagios.org/download/>
12. Move the tarball to the Nagios Master Server.
13. Extract the tarball:
`tar xzf nagios-plugins-version.tar.gz`
14. Move into the extracted tarball directory.
15. Execute:
`./configure`
16. Make the binaries:
`make`
17. Install the binaries: Either run `make install` or copy the binaries into the directory of your choosing inside the Nagios Installation Directory (typically, a folder called `libexec/`). The latter is the choice if you want more control over where your plug-in binaries go:
 - a. Alternatively, run `./configure -help` to see what other options are available for installation directories.

2.3 Fruity

Fruity is the configuration tool that simplifies the configuration of Nagios. Instead of being forced to edit text files to change Nagios's configuration, Fruity uses a MySQL database to store the configuration and uses a Web interface to make configuration easier. To use Fruity:

1. Download the latest Fruity distribution from (under **Downloads**):
<http://fruity.sourceforge.net/>
2. Move the tarball to the Nagios Master Server.
3. Extract the tarball:
`tar xzf fruity-version.tar.gz`
4. Copy the `fruity/` subdirectory to the Apache server's document root (find the document root in the Apache `httpd.conf` file).
5. Create a symlink or a copy of your Nagios's icon logo's directory to the Fruity's base directory with the name of `logos`.

2.4 NRPE

NRPE is the daemon that will be installed on all hosts that are monitored by Nagios. However, instead of installing NRPE on each host manually, install the daemon on the master

host in a folder that is different from the Nagios installation folder, because we provide scripts later that automate the installation process on all hosts. To install, follow these steps:

1. Download the latest NRPE distribution and the latest plug-in package from:
<http://www.nagios.org/download/>
2. Copy both tarballs to the Nagios Master Server and make sure that both files are owned by root.
3. Unpack the Nagios plug-ins package and enter the directory:

```
tar xzf nagios-plugins-*.tar.gz
cd nagios-plugins-*
```
4. Next, the plug-ins need to be compiled and installed. You *must* make sure that the installation directory is different from the Nagios installation directory. To do so, install to a master directory, such as /usr/nrpemaster. If you do not specify an installation directory, the installation will default to /usr/local/Nagios, which might be your current directory for the Nagios Master application.

```
./configure --prefix=/usr/nrpemaster --exec-prefix=/usr/nrpemaster
```
5. Next, make and then install the plug-ins:
 - a. **make**
 - b. **make install**
6. Now, unpack the NRPE package and enter the directory.

```
tar xzf nrpe-*.tar.gz
cd nrpe-*
```
7. Configure the NRPE installation. Currently, the NRPE configure script requires that the group Nagios exists. If it does not exist, create the group before moving to the next step:

```
groupadd nagios
./configure --prefix=prefix --exec-prefix=prefix --with-nrpe-user=someuser
--with-nrpe-group=somegroup
```

 - a. Replace *prefix* (for both prefix and exec-prefix) with the installation directory that you created in step 4 (/usr/nrpemaster)
 - b. Replace *someuser* with the name of the nagios user that you created earlier.
 - c. Replace *somegroup* with the name of the nagios group that you created earlier, not the group created in the previous step 1 (unless those groups are the same).
8. Run the install scripts.

```
make install-plugin
make install-daemon
make install-daemon-config
```
9. Next, assign the correct permissions to the plug-in directory and plug-ins:

```
chown -R NAGIOSUSER.NAGIOSGROUP /usr/nrpemaster
```
10. Install the nrpeinit script (see Appendix F, “NRPE scripts” on page 125 for usage tips) to /etc/init.d/nrpeinit and **symlink** it to /etc/init.d/rc3.d/S37nrpeinit:

```
ln -s /etc/init.d/nrpeinit /etc/init.d/rc3.d/S37nrpeinit
```



Apache WebServer, MYSQL, Nagios, Fruity, and NRPE configuration

Here are the steps to configure the Apache WebServer, MYSQL, Nagios, Fruity, and NRPE.

3.1 Configuring Apache to work with Nagios

There are a few steps to configure Apache to work with Nagios:

1. Open the file `httpd.conf` in the Apache2 installation for editing.
2. Make sure the server is listening on port 80: **Listen 80**
3. Find the line stating the Apache daemon user and group. Set the group to the Nagios group.
4. Create aliases for your Nagios HTML and CGI directories:
 - a. `<NAGIOS CGI DIRECTORY>` – Directory containing the Nagios CGI files
 - b. `<NAGIOS HTML DIRECTORY>` – Directory containing the Nagios HTML files
 - c. `<NAGIOS ETC DIRECTORY>` – Directory containing the Nagios Configuration files
 - d. `<HTTP CGI DIRECTORY>` – Apache Web directory for Nagios CGI files, for example:
<http://www.yourserver.com/nagios/cgi-bin>
 - e. `<HTTP HTML DIRECTORY>` – Apache Web directory for Nagios HTML files, for example:
<http://www.yourserver.com/nagios>

```
ScriptAlias <HTTP CGI DIRECTORY> <NAGIOS CGI DIRECTORY>
<Directory "<NAGIOS CGI DIRECTORY>">
    Options ExecCGI
    AllowOverride None
    Order allow,deny
    Allow from all
    AuthName "Nagios Access"
    AuthType Basic
    AuthUserFile <NAGIOS ETC DIRECTORY>/htpasswd.users
    Require valid-user
</Directory>

Alias <HTTP HTML DIRECTORY> <NAGIOS HTML DIRECTORY>
<Directory "<NAGIOS HTML DIRECTORY>">
    Options None
    AllowOverride None
    Order allow,deny
    Allow from all
    AuthName "Nagios Access"
    AuthType Basic
    AuthUserFile <NAGIOS ETC DIRECTORY>/htpasswd.users
    Require valid-user
</Directory>
```

3.2 MySQL

Set a password for the root user of MySQL:

```
mysqladmin -u root password NEWPASSWORD
```

3.3 Installing Fruity

To install Fruity:

1. Create a MySQL user for Fruity:

```
mysql -u root -p
```

2. Edit the file in the Fruity installation `sqldata/fruity-mysql.sql`. Add the line “CREATE DATABASE IF NOT EXISTS fruity;” to the top of the file.

3. Import the SQL data contained in `sqldata/fruity-mysql.sql`:

```
mysql -u root -p < sqldata/fruity-mysql.sql
```

4. Create a MySQL user for Fruity:

```
mysql -u root -p
GRANT ALL ON fruity.* TO '<FRUITY_DB_USER>'@'localhost' IDENTIFIED BY
'<FRUITY_DB_PASSWORD>'
```

5. Configure the file in Fruity's installation `/includes/config.inc`. Notes on the parameters are specified in the file.
6. Point your Web browser to your Fruity's installation on your Web server to verify that Fruity is working correctly.

3.4 Configuring Nagios using the Web interface

Now that Fruity is working, it is time to configure Nagios using the Web interface. There are other options for Nagios configurations than the one described here. Read the description of each setting as you set it to decide if each setting is appropriate for your environment. The steps are:

1. Navigate to Fruity's Web interface.
2. Enter the Main Config section:
 - a. Enter the Paths section. Fill in all the text boxes based on their descriptions:

Configuration directory	Directory containing the Nagios configuration files
Log file	nagios.log (typically in the Nagios log/ directory)
Object cache file	Objects.cache (typically in the Nagios var/ directory)
Temporary file	nagios.tmp (typically in the /tmp/ directory)
Status file	status.dat (typically in the Nagios var/ directory)
Log archive path	Archive/ (typically in the Nagios log/ directory)
Command file	rw/nagios.cmd (typically in the Nagios var/ directory)
Downtime file	Downtime.dat (typically in the Nagios var/ directory)
Comment file	Comments.dat (typically in the Nagios var/ directory)
Lock file	nagios.lock (typically in the Nagios var/ directory)
State retention file	retention.dat (typically in the Nagios var/ directory)
P1.pl file	p1.pl (typically the /usr/sbin/ directory)

- b. Enter the Status section:
 - i. Enable Aggregate Status Updates.
 - ii. Set the Aggregate Status Interval (typically 15 seconds).
- c. Enter the Security section. Set the user and group to the Nagios user and group you have been using all along.
- d. Enter the Restart Action section. Enable all settings.
- e. Enter the Logging section:
 - i. Set the Log Rotation Method to Daily.
 - ii. Enable all settings.
- f. Enter the External Commands section:
 - i. Enable External Commands.
 - ii. Set the check interval to -1.
 - iii. See the Nagios Documentation about External Commands for more explanation of this feature.
- g. Enter the Retention section:
 - i. Enable Retain State Information.
 - ii. Set the Retention Update Interval to 60.
 - iii. Enable Use Retained Program State.
 - iv. Disable Use Retained Scheduling Info.
- h. Enter the Intervals section:
 - i. Set the sleep time to .25.
 - ii. Set the Service Inter Check Delay Method to "s".
 - iii. Set the Max Service Check Spread to 30.
 - iv. Set the Host Inter Check Delay Method to "s".
 - v. Set the Max Host Check Spread to 30.
 - vi. Set the Service Interleave Factor to "s".
 - vii. Set the Maximum Concurrent Checks to 0.
 - viii. Set the Service Reaper Frequency to 0.
 - ix. Set the Interval Length to 60.
 - x. Disable Auto Reschedule Checks.
 - xi. Disable Aggressive Host Checking.
- i. Enter the Flap section. Disable Flap Detection.
- j. Enter the Timeouts section:
 - i. Set the Service Check Timeout to 60.
 - ii. Set the Host Check Timeout to 30.
 - iii. Set the Event Handler Timeout to 30.
 - iv. Set the Obsessive Compulsive Service Processor Timeout to 5.
 - v. Set the Performance Data Processor Command Timeout to 5.

- k. Enter the Obsess section:
 - i. Disable Obsess Over Service.
 - ii. Disable Obsess Over Hosts.
- l. Enter the Freshness section:
 - i. Enable Check Service Freshness.
 - ii. Set the Service Freshness Check Interval to 60.
- m. Enter the Other section:
 - i. Scroll to the Date Format. Choose a Date Format.
 - ii. Set “~!\$%^&*|”<>?,()=” for Illegal Object Name Characters.
 - iii. Set “~\$&|”<>” for Illegal Macro Output Characters.
 - iv. Set an Admin e-mail address.
 - v. Set an Admin Pager number.
- 3. Enter the CGI Config section:
 - a. Enter the Paths section:
 - i. Set the Physical HTML Path to the path to the Nagios HTML files.
 - ii. Set the URL HTML Path to the HTTP Nagios directory (typically /nagios).
 - b. Enter the Authentication section:
 - i. Enable Use Authentication.
 - ii. Authorize all actions with “nagios user”.
- 4. Enter the Resources section. Set \$USER1\$ to the absolute path of the Nagios plug-ins.
- 5. Enter the Export section and click **YES** to copy this configuration to Nagios.

You should at this point be able to navigate to Nagios in your Web browser, enter your specified user name and password, and see if the Web interface is working. If not, refer to the Nagios documentation for troubleshooting tips. Likely, it is a permissions problem or an authentication problem with the CGIs.

Note: Make sure Nagios is running.

All other configuration activities are described in the next section.

3.5 Configuring NRPE

The file `nrpe.cfg` governs all of the NRPE configuration. A sample configuration file is included with an NRPE installation. Configuring NRPE for your environment is as simple as editing this sample file:

1. Copy the sample config file to `nrpe.cfg`.
2. Set the `nrpe_user` to your “nagios user”.
3. Set the `nrpe_group` to your “nagios group”.
4. Add the Nagios Master Server IP address to the Allowed Host Address line.

Commands have been included in the sample file. You can simply comment these commands out and create similar commands that apply to the plug-ins that you have installed with NRPE. We explain creating commands in NRPE (and Fruity) in the next section.

After configuring NRPE on the Nagios Master Server, you must copy this installation and the new configuration to all of the hosts that use NRPE. Scripts are included in Appendix F, “NRPE scripts” on page 125 that you need to configure to your environment but will install NRPE easily on multiple hosts at a time.

To install NRPE on a new remote host for the first time, run the Install NRPE script. To refresh the configuration of NRPE, plug-in files, or binaries on a host that has already had NRPE installed, run the Reinstall NRPE script. This script does not overwrite state files or log files, so it preserves the state of any plug-ins that you use. To remove the NRPE installation from a remote host, run the Remove NRPE script. And to Start, Stop, or Restart the NRPE process, run the NRPE Manager script.

The scripts are governed by a file called `hostips`, which usually is located in the same folder as the scripts themselves. `Hostips` contains an IP address of any remote host for which you want to run the various scripts on each line. IP addresses can be commented out with a hash (#) symbol at the front of the line. Any IP address that is not commented out of this `hostips` file will be affected if you run any of these scripts. The scripts connect to each of the remote hosts specified in the file via `ssh` and perform the necessary tasks.

See Appendix F, “NRPE scripts” on page 125.



Nagios implementation and its plug-ins

This chapter gives you guidance about how to administer Nagios and its plug-ins. We include sample screen captures also to give you an idea of its capabilities.

4.1 Nagios

This section details the steps to setting up Nagios to check different aspects of your environment. Most of the work is done using Fruity. Nagios checks things in your environment based on hosts and services. *Hosts* are any devices that you can monitor with Nagios, which includes servers, disk controllers, switches, and even a Blue Gene rack. A *service* is an activity of a host that you want to monitor. This can be disk space, network reachability, processor load, specific users that have a failed login count over a certain number, or any Blue Gene jobs that have failed recently. You can check anything that can be scripted using Nagios. See 4.3, “Sample Nagios screen captures” on page 23 for illustrations of some of the capabilities of this solution.

4.1.1 Commands

Commands are the real strength of Nagios. *Commands* are all the things that Nagios uses to operate. Commands are the scripts that check the status of hosts, the status of services, send notifications, and execute event handlers. Basically, nearly everything done externally to Nagios needs to have a command specified to it.

If the plug-in will operate on the Nagios Master, create it using Fruity. If the plug-in will operate under NRPE, use the NRPE `nrpe.cfg` file that gets served to all NRPE hosts.

Using Fruity

Enter the Commands section and create a command. Enter an appropriate name and description. For the command line, `$USER1$` is a macro that is set to the plug-ins folder for the Nagios installation. You can view and edit this and other macros in the Resources section. You can also set dynamic arguments by specifying `$ARG1$`, `$ARG2$`, `$ARG3$`, and so forth as arguments that will change in your plug-in. These `$ARG*$` macros are set in specific Service definitions in the “Check Command Parameters” section. See 4.1.6, “Services” on page 19 for more information.

Using NRPE

Edit the file `nrpe.cfg`. It contains a list of commands that NRPE can run if Nagios calls them. To create your own commands, add a new line in this list of commands using a similar syntax to the other lines. Include the full path to the plug-in file (on the Remote NRPE installation) and any arguments that it needs. No dynamic command arguments (`$ARG1$` and so forth) are allowed in NRPE, so if you need more than one instance of a command, you need to create multiple lines in this file. To put this file and your finished plug-in on all NRPE hosts, run the NRPE reinstall script in the Installation section. An example of the NRPE `command.cfg` syntax is:

```
command[check_online_update]=/usr/local/ /nrpe/libexec/check_online_update.pl -w 1 -c 3
```

4.1.2 Time periods

Time periods are used in many areas of Nagios, including when to check a service, specifying when to send a notification, and so on. The Nagios documentation suggested that four time periods make a good base:

1. Create a new time period and name it `24x7`, give it a description, and enter `00:00-24:00` in all of the day fields.
2. Then, create a time period called `none` and leave all the day fields blank.

3. Then, create a time period called `workhours` with fields Monday-Friday containing 09:00-17:00 (or whatever applies in your environment).
4. Finally, create a time period called `nonworkhours` containing the opposite of the times listed in the `workhours` time period. Include 00:00-24:00 on weekends.

Now, create other time periods as you deem appropriate.

4.1.3 Contacts

A *contact* is an e-mail or pager listing for notification purposes. To create a new contact, navigate to Fruity, then to the Contacts section. You will see that our setup has two contact listings for each real person: one e-mail contact and one pager contact. In our example, the e-mail definition and the pager definition have separate settings. If you choose, you can create one contact with both an e-mail definition and a pager definition, but then both definitions use the same settings, so in case of notification, the contact will always get both an e-mail and a page. We decided that splitting it up into a separate e-mail contact and a separate pager contact gives you more control.

To create a new contact:

1. Enter an appropriate name and description. There is a problem with this form in Fruity, in that a single quotation mark (`'`), for instance, Matt's e-mail in the description, causes the creation of a new contact to fail. However, after you create a contact, you can edit it again and add back the single quotation mark that you were forced to remove.
2. You must also set a time period for which you will be contacted. If a host or service causes an error and it is not in within your specified time period, the notification will not be sent. See 4.1.2, "Time periods" on page 16.
3. Next, choose the host and service states about which you want to be notified:
 - The Unreachable state in a host means that in a hierarchical network, a host's parent host is down. Thus, the current host is not necessarily down, but it cannot be seen by Nagios.
 - A state of Unknown in a service means that the service is not necessarily in an error state, but that something is wrong with the plug-in, NRPE, or another area. Often, this can be from a configuration problem.
 - The Recovery state on both the host and the service means that it was previously in an error state and has returned to an OK state. See **Main Confide** → **Flap** for more information about *flapping*, which is disabled by default.
4. Choose a contact group with which to be associated. If you want to see the members of any group or create a new group, see the Contact Groups section. In a host or service, you specify contact groups to be notified instead of specific contacts, so your contact must be part of a contact group.
5. Finally, enter an e-mail address and a pager number, a mobile phone number, or both.
6. Now, click the new contact to bring it back up. Go to Notification Commands, which specifies what command you want to call when Nagios wants to send a notification to your contact. See the Nagios documentation for ideas about what to set up for notification commands. For service notifications, select only service commands. Select only host commands for host notifications. Add either an e-mail command or pager command based on the mode of communication for which you created this contact. If it is determined that a host or service notification needs to be sent, every command that you list in this section will be executed.
7. Remember to click **Export** and answer YES to export your changes to Nagios.

4.1.4 Templates

Templates are a simple way to apply a similar configuration to many hosts or services. Templates make it easy to make multiple changes or configurations and apply them to many similar hosts or services. For instance, you can have a template for Blue Gene Service Nodes that are all set to check specific services, as defined by that template. Also, you can create a template for services so that if you create a service that pings a host, you can create a ping template, and all of those similar services can use it. Using a template provides consistency and makes changes easier.

Host templates

A *host template* is a template of all of the settings and services of a certain type of host. For instance, it makes sense to create a template for all service node hosts. A template contains the base definition for that type of host, including the services that it will check. However, when setting up individual hosts, you can have them inherit all of the services and settings from a template but override some of the settings or add services that apply only to that individual host.

To create a new host template, navigate to Fruity. Click in the Templates section. Create a new template. You can even create a generic template so that any new host templates that you create can inherit all of the settings and services from the generic template. You can change settings universally this way, because a change to the generic template will propagate to all templates and therefore to all hosts. Name your template what you want and click into it after you create it.

You can add as much or as little definition in this template as you want. As a good base, we recommend that you enter the Checks section, enable passive checks, active checks, and the check command (the command to check if the host is alive – see 4.1.1, “Commands” on page 16) and set a number for the Maximum Check Attempts. In the Logging section, enable all the options. In the Notifications section, enable all of the notifications options (it is probably easier to change notification settings on a per contact basis than it is in a host template). Finally, add the host template to a contact group (see 4.1.3, “Contacts” on page 17). This is also be a good base for which to create a generic template. You can also add the template to a host group, which is the group of hosts in the Host Group view of Nagios. To create a new host group, enter the Host Group section. Add the template to a host group in the Group Membership section.

Finally, enter the Services section to create the services that you want to run on every host that will apply this template. See 4.1.6, “Services” on page 19 to learn how to create services.

Service templates

A *service template* is a base of configurations for a service. To create a service template, enter the Templates section. We recommend that you create a generic service template so that all services can inherit definitions from that one template. This makes them all easy to change. Service templates can inherit other service templates, and all the current service templates (and thus, all the services) can be under the generic service template. After you have created a generic service template, you can create other templates that inherit the generic template and you can expand upon the generic service template or override certain characteristics of the generic template. For instance, a ping service template is a specialized template that creates a ping service on any given host.

A generic service template (or all services) must contain a base set of definitions. In the Checks section, apply the Maximum Check Attempts, Normal Check Interval, Retry Check Interval, Active Checks (Enabled), Passive Checks (Enabled), Check Period (see 4.1.2, “Time periods” on page 16 to learn how to create different time periods), and Parallelize Checks

(Enabled). In the Logging section, enable all the options. In the Notifications section, enable all the options, selecting a 24x7 Notification Period and enabling all options. Similar to hosts, it is easier to set notification options on a per-contact basis than it is on a per-service basis.

Click **Add a New Service Template** to create a new service template. Create an appropriate name and apply a parent template if you want. We recommend that you apply a parent template, because you can always override or add new settings on top of a generic template. You can add as much or as little definition in this template as you want.

To apply your template to a service, create a new service inside a host and select your template from the Service Template drop-down box.

4.1.5 Hosts

A *host*, defined by Nagios, is a device that is being checked. Devices from front-end nodes to SAN controllers are considered hosts to Nagios.

To create a host, navigate to Fruity. Enter the Hosts section. Click to create a new host and enter a suitable name, description, and IP address. If you want, apply the host template of your choice. See “Host templates” on page 18 to create a host template.

After creating a host, enter the host. You can add as much or as little definition in this host as you want. If you are applying a template, you can override or add to any of the definitions that are already in place. As a good base, we recommend that you enter the Checks section. Enable passive checks, active checks, and the check command (“check-host-alive” by default) and set a number for Maximum Check Attempts. In the Logging section, enable all of the options. In the Notifications section, enable all of the notification options (it is probably easier to change notification settings on a per-contact basis than on a per-host basis). Finally, add the host template to a contact group. You can also add the host to a host group, which is the group of hosts in the Host Group view of Nagios. To create a new host group, enter the Host Group section. Add the host to a host group in the Group Membership section.

Finally, enter the Services section to create services that you want to run on every host that will apply this template. See 4.1.6, “Services” on page 19 to learn how to create services.

4.1.6 Services

A *service* is anything that is checked within a host. Commands and plug-ins are used with services. To create a service, you must have a host or host template for which to create the service. For a host, enter the Hosts section and select the host that you want. For a host template, select Templates and select the host template that you want. Enter the Services section. Create a new service and name it appropriately. If you want to apply a service template, choose one from the drop-down menu. See “Service templates” on page 18 to create a new service template.

We recommend that you apply at least a generic service template in order to keep all settings consistent across all services. You can override any of the definitions that are inherited by a service template in order to customize this new service. The important settings that you need to change are the Check Command setting under the Checks section and the Check Command Parameters section. If you use a service through NRPE, set Check Command to the command that was set up for the check_nrpe executable. Otherwise, choose a Check Command from the drop-down list, which are commands specified in the Commands section. See 4.1.1, “Commands” on page 16 if you need to create a new command for this service. After choosing a command, move to “Check Command Parameters.” If you have an NRPE command, the only parameter that you need to set is the command name found in the NRPE

nrpe.cfg file. For other commands, check the command definition in the Commands section to see how the parameters correspond to the command line.

Set a Service Group (if applicable). Finally, move to Export and select YES to copy your changes over to Nagios.

4.2 Plug-ins

Plug-ins are all of the external pieces to Nagios. Plug-ins are the scripts that Nagios uses to check hosts and services. Many plug-ins come installed in the Nagios Plug-ins package. This paper also includes many plug-ins for use with unique systems, such as Blue Gene and DB2® databases.

4.2.1 Development

Start by creating the script for the plug-in in your Nagios plug-in master directory. If the plug-in is supposed to be run from the Master Server by Nagios, it goes in the main Nagios Plug-ins folder. If the plug-in is meant to run on a remote host via NRPE, it goes in the NRPE installation folder on the Master Server. If both are true (for instance, the plug-in runs on the Nagios Master Server, which is a local plug-in, and on a Blue Gene Service Node, which is an NRPE plug-in), the finished plug-in must go in both folders.

Nagios has standards by which plug-ins must abide:

- ▶ Return values:
- ▶ 0 - OK state
- ▶ 1 - WARNING state
- ▶ 2 - CRITICAL state
- ▶ 3 - UNKNOWN state

Your plug-in must always return one of these states upon exit.

- ▶ Nagios will only process one line of output to standard out. If you output "Hello \n World", only Hello will appear in the Status Information section of Nagios.

If you need multiple lines of output, remember that Nagios is a Web interface and replacing "\n" with "
" will provide multiple line output in the Nagios Web interface.

If your plug-in builds a log file, and you want this log file to be viewable from the Web, see "Web-viewable log file" on page 21.

Next, you need to create a command for your plug-in file. See 4.1.1, "Commands" on page 16 for information about creating a command for this plug-in.

Finally, see section 4.1.6, "Services" on page 19 and point the Check Command to your new plug-in. If it is an NRPE plug-in, the Check Command is **check_nrpe** and the Check Command Parameters only have \$ARG1\$ as the command that you put in brackets in command.cfg. If appropriate, further down in the Checks section, remember to enable Event Handlers and choose the Event Handler command that you created to go along with this plug-in.

4.2.2 Resources

Next, we describe the available resources.

Event handlers

An *event handler* is a script that is set up to launch whenever a Nagios service changes state. Traditionally, the new state of the service is passed to the event handler so that the event handler can make decisions about what function to perform. However, an event handler can really do anything, because an event handler is not restricted by any rigid Nagios standards in the way that plug-ins are restricted.

First, create the script in the same folder as the Nagios plug-ins. A template already exists called `event_handler_template.pl` that we created for our specially built plug-ins that make use of HTML log files. See Appendix E, “Plug-in resources” on page 103 for this template.

When you have written the script, navigate to Fruity and enter the commands page. Create a new command, name it whatever makes sense to you, and enter the command line for executing the script, such as `$USER1$/my_event_handler.pl arg1 arg2 arg3`. Remember `$USER1$` is a macro that is defined in the Resources section of Fruity, which contains the Nagios plug-ins folder. Include arguments as needed. Nagios macros are useful for sending information to the script, such as the state of the service (`$SERVICESTATE$`) or the address of the host that contains the service (`$HOSTADDRESS$`). See Nagios documentation for macros for more information.

Save the command and move on to the service to which you want to add the event handler. You can add an event handler to a service or a service template. Enter the Checks section from the service that you select. Click to edit the Checks section and near the bottom are two changes that you need to make. Check the boxes to include Event Handler and Event Handler Command in the definition. Then, flip Event Handler to Enabled and change Event Handler Command to the command that you just created for your script.

Remember that the event handler is only launched when a service changes state. If a service is at a CRITICAL state with status information of Error, and the service checks again and the status information changes to Different Error!, that is not considered a state change in Nagios, because the service never changed from the CRITICAL state.

Web-viewable log file

To create a log file to which you can link and that you can view in Nagios, you must first edit your plug-in. At the end of any error state for which you want the log file visible (WARNING or CRITICAL), include an HTML link to the file's location, such as:

```
<br><a href='../logs/<HOST_IP>/<PLUGIN_NAME>/log.html'>
```

Click Here to view the log file.

Log files are put in `/srv/www/htdocs/nagios/logs/`, and they go in a folder specified by a unique host identifier. In most cases, this is the host's IP address. However, in the case of Simple Network Management Protocol (SNMP) hosts, such as DS4000™, the host name of the device is used. Then, they go in a folder specified by the plug-in name. For example, the log file for BG/L Link Cards on BGLSN goes in:

`/srv/www/htdocs/nagios/logs/172.22.60.254/check_bg_linkcard/linkcard.log`. The IP address was chosen as a host identifier to include information dynamically in a script. So, you can get the IP address with the `hostname -i` command and dynamically insert that IP address into your HTML link.

After you generate your log file, you need to copy it into the appropriate folder in `/srv/www/htdocs/nagios`. If the plug-in is running on a remote host, the log file needs to be SCPed to bg-management and copied into that folder. Either way, be sure to set the correct owner (`nagios.nagcmd`) and permissions (775) for the file and the directory if you create the directory. If you do not set the correct owner and permissions, the Web server cannot access

the log file. Example 4-1 shows a sample of the commands that scp, copy, and edit the file from the remote host to the Nagios Master Server.

Example 4-1 Sample of the commands that scp, copy, and edit the file from the remote host to Nagios

```
#SCP the Log File to the correct place on the Nagios Master Server, and set the
permissions correctly.
#YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is to send a
copy of the log file to the Nagios master server, put it in a folder under the
Apache Web Directory, then change the permissions and the owner to be the nagios
user and nagios group.
system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@${nagios_ip}:~USERID' >
/dev/null");

system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no \"sudo
mkdir -p $master_log_loc \"' > /dev/null");

system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no \"sudo
chmod 775 $master_log_loc \"' > /dev/null");

system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no \"sudo
chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");

system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no \"sudo
cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");

system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no \"sudo
chmod 775 $master_log_loc/$logfile \"' > /dev/null");

system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no \"sudo
chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null");
```

When this is ready, create an event handler (see “Event handlers” on page 21). You can simply edit the template file available. Notice that the template file calls a script called `build_html.pl` on any warning or critical state. This builds an HTML file for viewing the log file and puts the HTML file in the same folder as the log file. Notice the arguments of the event handler template. Set those up in Fruity using macros, as described in the file’s contents and in the event handler tutorial.

If you set everything up correctly (and the permissions are correct), every time that your plug-in now enters an error state, the event handler will be launched and will call the `build_html` script, and you will find an HTML file linked via your plug-in. The HTML file will display a log file of the error. If this process does not work, you might have a problem with permissions, or one of the scripts failed in some way. Nagios cannot report a failure in an event handler, so you might have to run the script manually to discover the problem.

4.2.3 Included plug-ins

Several plug-ins are included with this paper. These plug-ins check several types of unique devices, such as a Blue Gene/L™ Supercomputer, a Blue Gene/P™ Supercomputer, and DB2 databases. Also, included with this paper are resources for these plug-ins, such as event handlers and a script that builds HTML files for viewing log files in Nagios. See Part 1, “Appendixes” on page 29 for the code to the plug-ins.

4.3 Sample Nagios screen captures

This section shows several of the capabilities and information available using Nagios. Figure 4-1 shows a status summary for the system administrator of the systems that are being monitored.

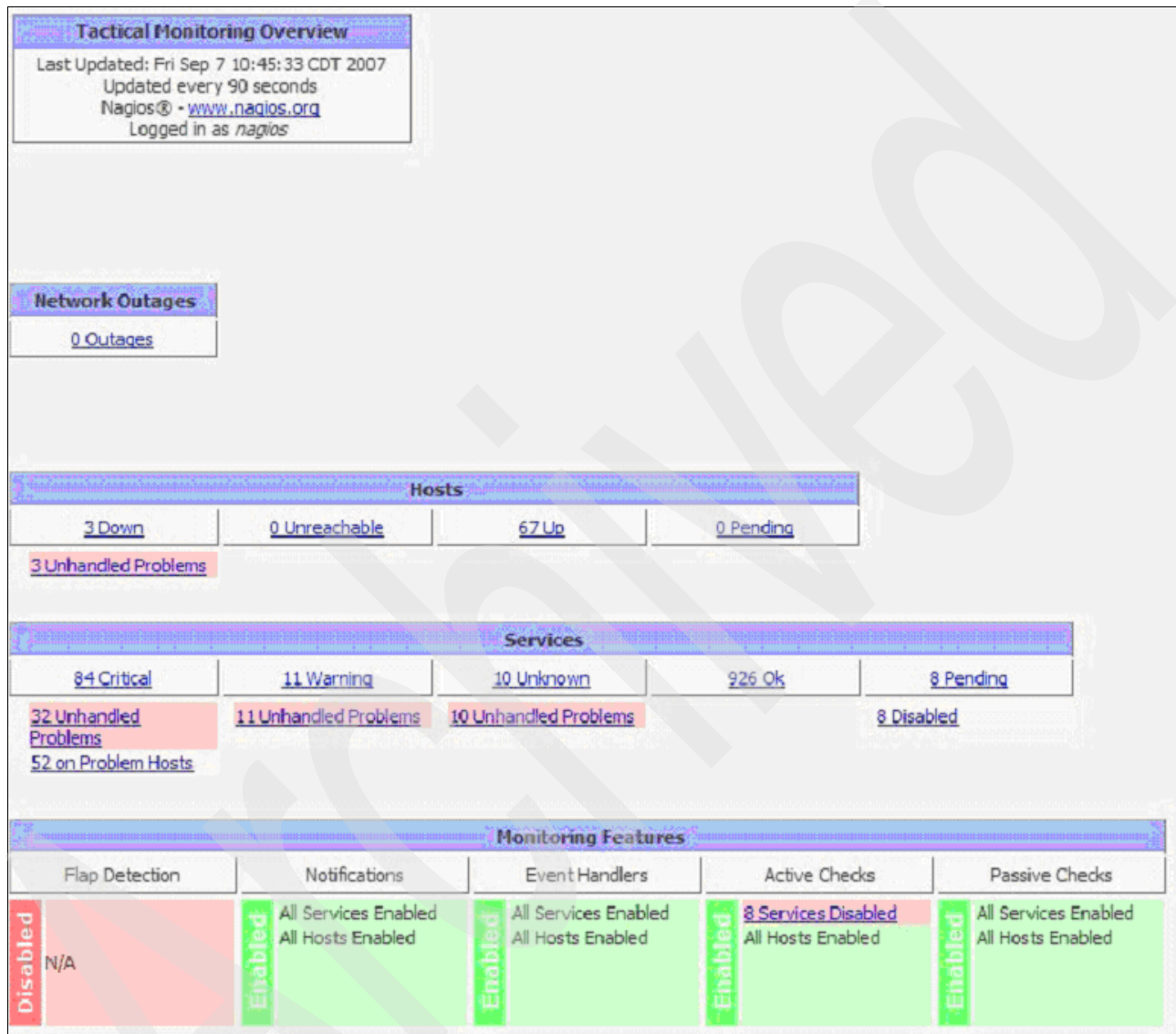


Figure 4-1 Tactical monitoring overview

Figure 4-2 on page 24 shows the system administrator which hosts are up and communicating in the network. This screen capture helps system administrators quickly identify any hosts that are in a down state. Clicking on a host allows the system administrator to drill down into the details of that system.

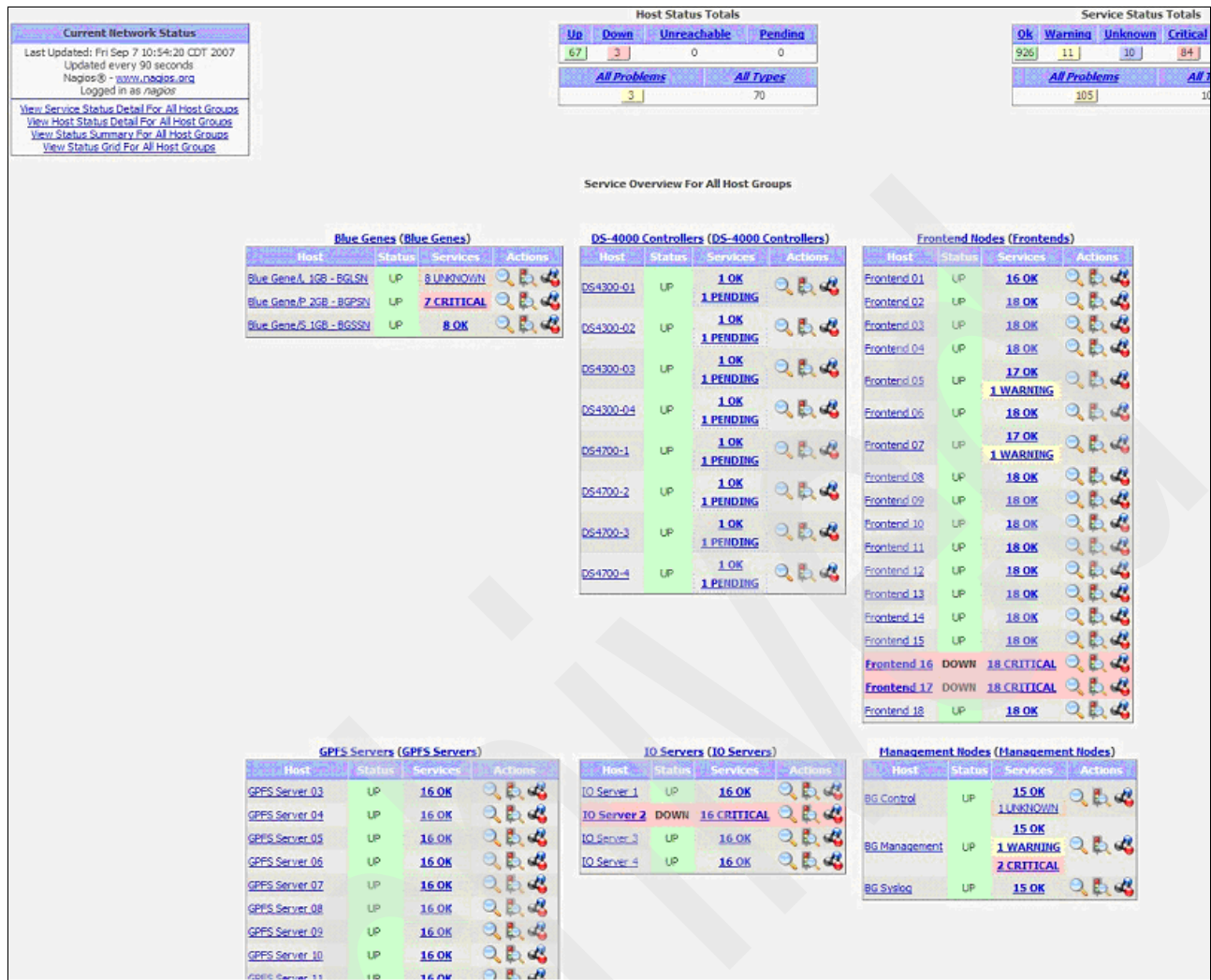


Figure 4-2 Current network status

Nagios allows you to define host groups in order to logically organize them by function, as shown in Figure 4-3 on page 25. Clicking on the group title above the group box displays summary information for that group. Clicking on an individual host, or on a service link, gives you detailed information about that host or that service link.





























































Service Overview For All Host Groups											
Blue Genes (Blue Genes)				DS-4000 Controllers (DS-4000 Controllers)				Frontend (Frontend)			
Host	Status	Services	Actions	Host	Status	Services	Actions	Host	Status	Services	Actions
Blue Gene/L 1GB - PSN	UP	7 OK 1 WARNING	  	DS4300-01	UP	1 OK 1 PENDING	  	Frontend 01	UP	1 OK 1 PENDING	  
Blue Gene/P 2GB - PSN	UP	1 OK 1 WARNING 5 CRITICAL	  	DS4300-02	UP	1 OK 1 PENDING	  	Frontend 02	UP	1 OK 1 PENDING	  
Blue Gene/S 1GB - PSN	UP	8 OK	  	DS4300-03	UP	1 OK 1 PENDING	  	Frontend 03	UP	1 OK 1 PENDING	  
				DS4300-04	UP	1 OK 1 PENDING	  	Frontend 04	UP	1 OK 1 PENDING	  
				DS4700-1	UP	1 OK 1 PENDING	  	Frontend 05	UP	1 OK 1 PENDING	  
				DS4700-2	UP	1 OK 1 PENDING	  	Frontend 06	UP	1 OK 1 PENDING	  
				DS4700-3	UP	1 OK 1 PENDING	  	Frontend 07	UP	1 OK 1 PENDING	  
				DS4700-4	UP	1 OK 1 PENDING	  	Frontend 08	UP	1 OK 1 PENDING	  
								Frontend 09	UP	1 OK 1 PENDING	  

Figure 4-3 Nagios Sysadmin status dashboard

When you click on the host, the page shown in Figure 4-4 on page 26 lists the status of all monitored services on that individual host. You can also use the navigator pane on the left or click on the available hot links on the page to display details. Clicking on any underscored link allows you to drill down into more detail.

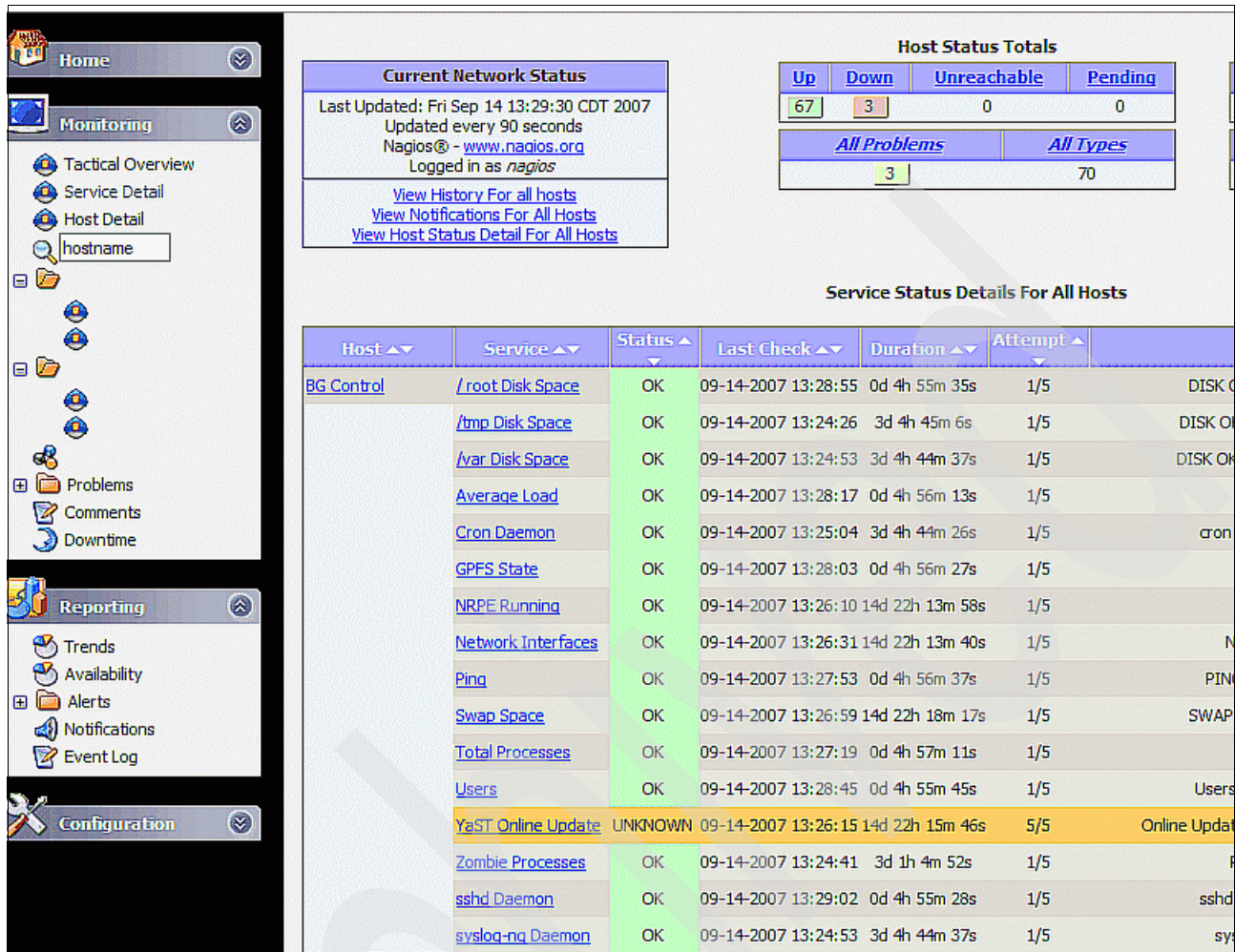


Figure 4-4 Nagios service dashboard

Clicking on an individual service link drills down to the detail about the service state. For example, the disk space status is shown in Figure 4-5 on page 27. The screen capture shows whether the alert threshold has been exceeded. Green shading indicates an OK condition, while yellow and red shading indicate that attention is needed.

Within this page, in the service commands box at the right, you can disable the status checking for maintenance purposes.

Service Information

Last Updated: Fri Sep 14 13:31:35 CDT 2007
Updated every 90 seconds
Nagios® - www.nagios.org
Logged in as *nagios*

[View Information For This Host](#)
[View Status Detail For This Host](#)
[View Alert History For This Service](#)
[View Trends For This Service](#)
[View Alert Histogram For This Service](#)
[View Availability Report For This Service](#)
[View Notifications For This Service](#)

Service

/ root Disk Space

On Host

Control Server for Blue Gene
(BG Control)

Member of

Disk Space

172.22.80.50

Service State Information

Current Status:

OK

Status Information:

DISK OK - free space: / 7984 MB (97%
inode=-):

Performance Data:

/=212MB;6556;7376;0;8196

Current Attempt:

1/5

State Type:

HARD

Last Check Type:

ACTIVE

Last Check Time:

09-14-2007 13:28:55

Status Data Age:

0d 0h 2m 40s

Next Scheduled Active Check:

09-14-2007 13:33:55

Latency:

0.103 seconds

Check Duration:

0.050 seconds

Last State Change:

09-14-2007 08:33:55

Current State Duration:

0d 4h 57m 40s

Last Service Notification:

N/A

Current Notification Number:

0

Is This Service Flapping?

N/A

Percent State Change:

N/A

In Scheduled Downtime?

NO

Last Update:

09-14-2007 13:31:28

Active Checks:

ENABLED

Passive Checks:

ENABLED

Obsessing:

ENABLED

Notifications:

ENABLED

Event Handler:

ENABLED

Flap Detection:

ENABLED

Service Commands

[Disable active checks of this serv](#)
[Re-schedule the next check of this serv](#)
[Submit passive check result for this serv](#)
[Stop accepting passive checks for this serv](#)
[Stop obsessing over this serv](#)
[Disable notifications for this serv](#)
[Schedule downtime for this serv](#)
[Disable event handler for this serv](#)
[Disable flap detection for this serv](#)

Service Comments

[Add a new comment](#)
[Delete all comments](#)

Entry Time	Author	Comment	Comment ID	Persistent	Type	Expires
This service has no comments associated with it						

Figure 4-5 Nagios service drilldown

Nagios offers various ways to view the status information. Figure 4-6 on page 28 shows this information in a report-style format. By clicking on any hot link, as described before, you can drill down into the system details.

In summary, Nagios provides an easy-to-use graphical interface to monitor host systems and status details.





























































Blue Genes (Blue Genes)		
Host	Services	Actions
Blue Gene/L 1GB - BGLSN	BG/L Job History BG/L Link Cards BG/L Midplanes BG/L Node BG/L Node Cards BG/L Processor Cards BG/L Service Actions BG/L Service Cards	  
Blue Gene/P 2GB - BGPSN	BG/P Job History BG/P Link Cards BG/P Midplanes BG/P Node Cards BG/P Nodes BG/P Service Actions BG/P Service Cards	  
Blue Gene/S 1GB - BGSSN	BG/S Job History BG/S Link Cards BG/S Midplanes BG/S Node Cards BG/S Nodes BG/S Processor Cards BG/S Service Actions BG/S Service Cards	  
DS-4000 Controllers (DS-4000 Controllers)		
Host	Services	Actions
DS4300-01	Ping SNMP Health Critical	  
DS4300-02	Ping SNMP Health Critical	  
DS4300-03	Ping SNMP Health Critical	  
DS4300-04	Ping SNMP Health Critical	  
DS4700-1	Ping SNMP Health Critical	  
DS4700-2	Ping SNMP Health Critical	  
DS4700-3	Ping SNMP Health Critical	  
DS4700-4	Ping SNMP Health Critical	  
Frontend Nodes (Frontends)		
Host	Services	Actions
Frontend 01	/root Disk Space /tmp Disk Space /var Disk Space Average Load Cron Daemon GPFS State NRPE Daemon Network Interfaces Ping Swap Space Total Processes Users YaST Online Update Zombie Processes sshd Daemon syslog-ng Daemon	  
Frontend 02	/root Disk Space /FS Disk Space /tmp Disk Space /var Disk Space Average Load Cron Daemon GPFS State NFS Exports NRPE Daemon Network Interfaces Ping Swap Space Total Processes Users YaST Online Update Zombie Processes sshd Daemon syslog-ng Daemon	  
Frontend 03	/root Disk Space /FS Disk Space /tmp Disk Space /var Disk Space Average Load Cron Daemon GPFS State NFS Exports NRPE Daemon Network Interfaces Ping Swap Space Total Processes Users YaST Online Update Zombie Processes sshd Daemon syslog-ng Daemon	  
Frontend 04	/root Disk Space /FS Disk Space /tmp Disk Space /var Disk Space Average Load Cron Daemon GPFS State NFS Exports NRPE Daemon Network Interfaces Ping Swap Space Total Processes Users YaST Online Update Zombie Processes sshd Daemon syslog-ng Daemon	  
Frontend 05	/root Disk Space /FS Disk Space /tmp Disk Space /var Disk Space Average Load Cron Daemon GPFS State NFS Exports NRPE Daemon Network Interfaces Ping Swap Space Total Processes Users YaST Online Update Zombie Processes sshd Daemon syslog-ng Daemon	  
Frontend 06	/root Disk Space /FS Disk Space /tmp Disk Space /var Disk Space Average Load Cron Daemon GPFS State NFS Exports NRPE Daemon Network Interfaces Ping Swap Space Total Processes Users YaST Online Update Zombie Processes sshd Daemon syslog-ng Daemon	  
Frontend 07	/root Disk Space /FS Disk Space /tmp Disk Space /var Disk Space Average Load Cron Daemon GPFS State NFS Exports NRPE Daemon Network Interfaces Ping Swap Space Total Processes Users YaST Online Update Zombie Processes sshd Daemon syslog-ng Daemon	  
Frontend 08	/root Disk Space /FS Disk Space /tmp Disk Space /var Disk Space Average Load Cron Daemon GPFS State NFS Exports NRPE Daemon Network Interfaces Ping Swap Space Total Processes Users YaST Online Update Zombie Processes sshd Daemon syslog-ng Daemon	  
Frontend 09	/root Disk Space /FS Disk Space /tmp Disk Space /var Disk Space Average Load Cron Daemon GPFS State NFS Exports NRPE Daemon Network Interfaces Ping Swap Space Total Processes Users YaST Online Update Zombie Processes sshd Daemon syslog-ng Daemon	  

Figure 4-6 Nagios status grid display



Part 1

Appendixes

Soft copies of these scripts are available online in zipped TGZ format at:

<ftp://www.redbooks.ibm.com/redbooks/REDP4356>

Archived

Blue Gene/L plug-ins

These are database scripts for running Nagios on a Blue Gene (BG) system.

Soft copies of these scripts are available online in zipped TGZ format at:

<ftp://www.redbooks.ibm.com/redbooks/REDP4356>

A.1 Check BG/L jobs

This script checks the Blue Gene DB2 database. It queries table TBGLJOB_HISTORY for any entries with an exit status that is not equal to 0. It will return a warning status (exit status 1) and give a message indicating which job IDs experienced errors. It will also write to a log file called joberror.log. In order to not continue returning the same errored job, two files will keep a record of the last errored job found (jobid.error) and the last errored job cleared by the system administrators through Nagios (jobid.clear).

Additional requirements

You must run the:

- ▶ Check BG Jobs Event Handler script
- ▶ Build Log HTML script

When run in Nagios, Check BG Jobs will never return to an OK state from a WARNING state automatically. The user or system administrator must manually submit a “Passive Service Check Result” via Nagios. You can do this in one of two ways. This is where the two “Additional Requirements” scripts just listed come into play; make sure they are properly installed and configured in Nagios before proceeding.

When a job error is presented in Nagios, the status information section will link to a log file. At the top of this page, there is a text box and a Submit button. Enter the text that you want to see in the Status Information section and click Submit. This will clear the log file and set the state to “OK” in Nagios. Note that this action causes Nagios to ignore any previous job failure. If Check BG Jobs reports an error again, it will be because a job has failed since the last time that you submitted this OK state. Basically, use the text box and button to clear any errors you have disregarded or already handled.

This script is meant to be run by Nagios. The sysadmin alerting and notification script is taken care of by Nagios. This script only checks the database and reports its findings.

There are several variables at the top of the script that have been hard-coded, which include IP addresses and paths to file locations. Change these variables to match the environment:

nagios_ip	The IP address of the Nagios master server
master_log_loc	The location of the directory on the master host where the log file will be stored (in order to view it using the Nagios Web interface)
local_log_loc	The location of the directory on the remote host where the log file is stored
logfile	The name of the log file
loglink	The HTML link (and text) to the log file that will be viewed using the Nagios Web interface. This will be added to the end of all error-state output.

Other things to check

Check to:

- ▶ Make sure the paths to jobid.error and jobid.clear are correct.
- ▶ Make sure the DB2 Connection string (found on the “my \$dbh = DBI->connect...” line) has the correct database name, database user, and database password.
- ▶ Make sure the SQL query (found on the “my \$sql = ...” line) has the correct arguments and, most importantly, queries the correct table name.

Usage

The usage command is:

```
./check_bg1_jobs.pl
```

Return value

The return value is either:

- ▶ Return 0 (OK state in Nagios) if no new failed jobs are found since the last “OK” by the user
- ▶ Return 1 (WARNING state in Nagios) if one or more new failed jobs are found since the last “OK” by the user

Code

Example A-1 is the code to check BG/L jobs.

Example: A-1 Check BG/L jobs

```
#!/usr/bin/perl
###
#####
###
### check_bg1_jobs.pl
### Created 6/20/2007
###
### Checks the Blue Gene DB2 database. Queries table TBGLJOB_HISTORY for any
### entries with exit status not equal to 0. It will return a warning status (exit
### status 1) and give a message of what job IDs have errored. It will also write
### to a log file, located at /var/log/bgmr/nagios/check_bg_jobs/joberror.log. In
### order to not keep returning the same errored job, two files will keep a record
### of the last errored job found (/var/lib/bgmr/nagios/check_bg_jobs/jobid.err)
### and the last errored job cleared by the system administrators via Nagios
### (/var/lib/bgmr/nagios/check_bg_jobs/jobid.clean).
###
### This script is meant to be run by Nagios. Sysadmin
### alerting and notification is taken care of via Nagios.
### This script only checks the database and reports its
### findings.
###
#####
#Nagios Master Server
my $nagios_ip = "IP ADDRESS HERE";
#This Host's Linux Hostname
my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#Location of Web-viewable log file on the Nagios Master Server
my $master_log_loc = "WEB DIRECTORY HERE";
#Location of local log file on remote host
my $local_log_loc = "LOG DIRECTORY HERE";
#Name of the Log File
my $logfile = "joberrors.log";
#HTML Link to add to the end of all errored output
my $loglink = "<br><a href='WEB URL HERE/joberrors.html'>Click Here</a> to
view the log file.";
use DBI;
use DBD::DB2::Constants;
```

```

        use DBD::DB2;
        use strict;
        # Nagios specific
        my $TIMEOUT = 15;
        my %ERRORS=( 'OK'=>0, 'WARNING'=>1, 'CRITICAL'=>2, 'UNKNOWN'=>3,
'DEPENDENT'=>4);
#       #DB2 Sourcing
#       my $db2result;
#       $db2result = `./discovery/db.src`;
        #DB2 Connection String
        my $dbh = DBI->connect("dbi:DB2:DB_NAME", "DB_USER", "DB_PASSWORD");
        #If DB2 connection fails, print error and send CRITICAL ERROR
        if(!(defined $dbh))
        {
            print "CRITICAL : Error Connecting to DB2 Database:
$DBI::errstr\n";
            exit $ERRORS{'CRITICAL'};
        }
#Read the JobID of the last cleared job.
        open(CLEANIDINFILE, "<STATE_FILE_DIRECTORY_HERE/jobid.clear");
        my @file = <CLEANIDINFILE>;
        my $cleanmaxid = $file[0];
        chomp($cleanmaxid);
        close(CLEANIDINFILE);
        #Read the JobID of the last job found.
        open(ERRIDINFILE, "<STATE_FILE_DIRECTORY_HERE/jobid.error");
        my @file = <ERRIDINFILE>;
        my $errmaxid = $file[0];
        chomp($errmaxid);
        close(ERRIDINFILE);
        #SQL Query
        my $sql = "select jobid, username, jobname, blockid, exitstatus, errtext,
executable, args, char(entrydate) from tbgljob_history where exitstatus<>0 and
jobid > $cleanmaxid order by jobid asc";
        my $sth = $dbh->prepare($sql);
        $sth->execute();
        my( $jobid, $username, $jobname, $blockid, $exitstatus, $errtext,
$executable, $args, $entrydate);
        my ($output, $fileoutput);
        my $time;
        my $date;
        my $shortoutput;
        my $count = 0;
        my $maxid = 0;

        #Fetch DB2 data row by row
        while( ( $jobid, $username, $jobname, $blockid, $exitstatus, $errtext,
$executable, $args, $entrydate) = $sth->fetchrow() )
        {
            $count = $count + 1;
            #Remove Leading & Trailing White Space
            $jobid =~ s/^\s+//;
            $jobid =~ s/\s+$//;
            $username =~ s/^\s+//;
            $username =~ s/\s+$//;

```

```

$jobname =~ s/^\s+//;
$jobname =~ s/\s+$//;
$exitstatus =~ s/^\s+//;
$exitstatus =~ s/\s+$//;
$blockid =~ s/^\s+//;
$blockid =~ s/\s+$//;
$errtext =~ s/^\s+//;
$errtext =~ s/\s+$//;
$executable =~ s/^\s+//;
$executable =~ s/\s+$//;
$args =~ s/^\s+//;
$args =~ s/\s+$//;
$entrydate =~ s/^\s+//;
$entrydate =~ s/\s+$//;
$shortoutput .= "$jobid, ";
#Split EntryDate into Date and Time
$entrydate =~ m/(\d+-\d+-\d+)-(\d+\.\d+\.\d+)/;
$date = $1;
$time = $2;
$output .= "JOB $jobid FAILED $date $time: User: $username, Name:
$jobname, Exit: $exitstatus, Error: $errtext, Block: $blockid Exe (with args):
${executable} ${args} <br><br> ";
#If JobID is greater than the last JobID found.
if($jobid > $errmaxid)
{
    $fileoutput .= "JOB $jobid FAILED $date $time: User:
$username, Name: $jobname, Exit: $exitstatus, Error: $errtext, Block: $blockid \n
Exe (with args): ${executable} ${args}\n\n";
    $maxid = $jobid;
}
}

#Disconnect from DB2
$sth->finish();
$dbh->disconnect();
#Set a new max JobID, write it to the file.
if ($maxid != 0)
{
    $errmaxid = $maxid;
}
open(IDOUTFILE, ">STATE_FILE_DIRECTORY_HERE/jobid.error");
print IDOUTFILE $errmaxid;
close(IDOUTFILE);

#If there are errors to print, write them to the log file.
if( $fileoutput ne "")
{
    open(JOBLOGFILE, ">>$local_log_loc/$logfile");
    print JOBLOGFILE $fileoutput;
    close(JOBLOGFILE);
}
#SCP the Log File to the correct place on the Nagios Master Server,
#and set the permissions correctly.

#YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is

```

```

        #to send a copy of the log file to the Nagios master server, put it in a
        #folder under the Apache Web Directory, then change the permissions
        #and the owner to be the nagios user and nagios group.
        system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@${nagios_ip}:~USERID' >
/dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo mkdir -p $master_log_loc \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc/$logfile \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null");
        $shortoutput = substr($shortoutput, 0, - 2);

        #If length of short output is longer than Nagios allows,
        #Shorten it further.
        if( (length("$count Jobs Failed: ") + length(".$loglink") +
length($shortoutput)) > 230 )
        {
                $shortoutput = "$count Jobs Failed.$loglink";
        }
        else
        {
                $shortoutput = "$count Jobs Failed: $shortoutput.$loglink";
        }
        if( $count > 0 )
        {
                #If full output is longer than Nagios allows,
                #Use the short output.
                if( length($output) + length($loglink) > 320 )
                {
                        $output = $shortoutput;
                }
                else
                {
                        $output = substr($output, 0, - 9);
                        $output .= $loglink;
                }
                print( $output);
                exit $ERRORS{"WARNING"};
        }
        else
        {
                print "OK : No Failed Jobs\n";
                exit $ERRORS{"OK"};
        }

```

A.2 Check BG/L link card

This script checks the Blue Gene DB2 database. It queries table TBGLLINKCARD for any entries with status not equal to "A". It will return a critical status (exit status 2) and give a message indicating which link cards have an error. It will also write to a log file that is located at `var/log/bgmr/nrpe/check_bg_linkcard/linkcard.log`. If the problem is resolved and a link card's status changes back to "A", the script will return OK status (exit status 0) and the log file will clear itself.

Optional additions

You can optionally run the:

- ▶ Check BG Link Card Event Handler script
- ▶ Build Log HTML script

This script is meant to be run by Nagios. The sysadmin alerting and notification script is taken care of via Nagios. This script only checks the database and reports its findings.

There are several variables at the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

nagios_ip	The IP address of the Nagios master server
master_log_loc	The location of the directory on the master host where the log file will be stored (in order to view it using the Nagios Web interface)
local_log_loc	The location of the directory on the remote host where the log file is stored
logfile	The name of the log file
loglink	The HTML link (and text) to the log file that will be viewed using the Nagios Web interface. This will be added to the end of all error-state output.

Other things to check

Check to:

- ▶ Make sure the DB2 Connection string (found on the "my \$dbh = DBI->connect..." line) has the correct database name, database user, and database password.
- ▶ Make sure the SQL query (found on the "my \$sql = ..." line) has the correct arguments and, most importantly, queries the correct table name.

Usage

The usage command is:

```
./check_bg1_linkcard.pl
```

Return value

The return value is either:

- ▶ Return 0 (OK state in Nagios) if all link cards have a status of "A"
- ▶ Return 2 (CRITICAL state in Nagios) if any link cards have a status that is not "A"

Code

Example A-2 on page 38 is the code to check the BG/L link card.

Example: A-2 Check BG/L link card

```
#!/usr/bin/perl
###
#####
###
### check_bg_l_linkcard.pl
### Created 6/25/2007
###
### Checks the Blue Gene DB2 database. Queries table TBGLLINKCARD for any entries
### with status not equal to 'A'. It will return a critical status (exit status
### 2) and give a message of what Link Cards have errored. It will also write to a
### log file, located at /var/log/bgmr/nrpe/check_bg_linkcard/linkcard.log. If the
### problem is resolved, and a Link Card's status changes back to 'A', then the
### script will return OK status (exit status 0) and the log file will clear
### itself.
###
### This script is meant to be run by Nagios. Sysadmin alerting and notification
### is taken care of via Nagios. This script only checks the database and reports
### its findings.
###
#####
#Nagios Master Server
my $nagios_ip = "IP ADDRESS HERE";
#This Host's Linux Hostname
my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#Location of Web-viewable log file on the Nagios Master Server
my $master_log_loc = "WEB DIRECTORY HERE";
#Location of local log file on remote host
my $local_log_loc = "LOG DIRECTORY HERE";
#Name of the Log File
my $logfile = "linkcard.log";
#HTML Link to add to the end of all errored output
my $loglink = "<br><a href='WEB URL HERE/linkcard.html'>Click Here</a> to
view the log file.";

use DBI;
use DBD::DB2::Constants;
use DBD::DB2;
use strict;
my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#For Nagios
my $TIMEOUT = 15;
my
%ERRORS=('OK'=>0,'WARNING'=>1,'CRITICAL'=>2,'UNKNOWN'=>3,'DEPENDENT'=>4);
#
# DB2 Sourcing
#
my $db2result;
#
$db2result = `./discovery/db.src`;
#DB2 Connection String
my $dbh = DBI->connect("dbi:DB2:DB_NAME", "DB_USER", "DB_PASSWORD");
#If DB2 connection fails, print error and send CRITICAL ERROR
if(!(defined $dbh))
{
```



```

        print "CRITICAL : Error Connecting to DB2 Database:
$DBI::errstr\n";
        exit $ERRORS{'CRITICAL'};
    }
    #SQL Query
    my $sql = "select serialnumber, midplaneserialnumber, posinmidplane,
status, location from tbgl1linkcard where status <> 'A' and status <> 'M' order by
serialnumber asc";
    my $sth = $dbh->prepare($sql);
    $sth->execute();
    my ($serialnumber, $midplaneserialnumber, $posinmidplane, $status,
$location);
    my $count = 0;
    my ($output, $shortoutput);
    my ($serialleft, $serialright);
    #Fetch DB2 data row by row
    while ( ($serialnumber, $midplaneserialnumber, $posinmidplane, $status,
$location) = $sth->fetchrow() )
    {
        $count = $count + 1;
        #Remove Leading & Trailing White Space
        $serialnumber =~ s/^\s+//;
        $serialnumber =~ s/\s+$//;
        $midplaneserialnumber =~ s/^\s+//;
        $midplaneserialnumber =~ s/\s+$//;
        $posinmidplane =~ s/^\s+//;
        $posinmidplane =~ s/\s+$//;
        $status =~ s/^\s+//;
        $status =~ s/\s+$//;
        $location =~ s/^\s+//;
        $location =~ s/\s+$//;
        $output .= "Link Card failed: Pos=$posinmidplane Status=$status
Location=$location <br><br> ";
    }
    $sth->finish();
    $dbh->disconnect();
    #If there are errors to print, write them to the log file.
    open(MYOUTFILE, ">/var/log/bgmr/nrpe/check_bg_linkcard/linkcard.log");
    my $fileoutput = $output;
    $fileoutput =~ s/<br><br>/\n/g;
    print MYOUTFILE $fileoutput;
    close(MYOUTFILE);
    #SCP the Log File to the correct place on the Nagios Master Server,
    #and set the permissions correctly.

    #YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is
    #to send a copy of the log file to the Nagios master server, put it in a
    #folder under the Apache Web Directory, then change the permissions
    #and the owner to be the nagios user and nagios group.
    system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@${nagios_ip}:~USERID' >
/dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo mkdir -p $master_log_loc \"' > /dev/null");

```

```

        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc/$logfile \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null");
        #If there are errors, print them and send a CRITICAL status.
        if ($output ne "")
        {
            #If full output is longer than Nagios allows,
            #Use the short output.
            if( length($output) + length($loglink) > 320 )
            {
                $output = "$count Link Cards Failed. $loglink";
            }
            else
            {
                $output = substr($output, 0, - 9);
                $output .= $loglink;
            }
            print( $output, "\n" );
            exit $ERRORS{"CRITICAL"};
        }
        else
        {
            print "OK : No Link Cards Failed\n";
            exit $ERRORS{"OK"};
        }
    }
}

```

A.3 Check BG/L midplane

This script checks the Blue Gene DB2 database. It queries table TBGLMIDPLANE for any entries with status not equal to "A". It will return a critical status (exit status 2) and give a message indicating which midplanes have an error. It will also write to a log file, which is located at /var/log/bgmr/nrpe/check_bg_linkcard/midplane.log. If the problem is resolved and a midplane's status changes back to "A", the script will return OK status (exit status 0) and the log file will clear itself.

Optional additions

You can also run the:

- ▶ Check BG Midplane Event Handler script
- ▶ Build Log HTML script

This script is meant to be run by Nagios. The sysadmin alerting and notification script is taken care of via Nagios. This script only checks the database and reports its findings.

There are several variables at the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

nagios_ip	The IP address of the Nagios master server
master_log_loc	The location of the directory on the master host where the log file will be stored (in order to view it using the Nagios Web interface)
local_log_loc	The location of the directory on the remote host where the log file is stored
logfile	The name of the log file
loglink	The HTML link (and text) to the log file that will be viewed using the Nagios Web interface. This will be added to the end of all error-state output.

Other things to check

Check to:

- ▶ Make sure the DB2 Connection string (found on the “my \$dbh = DBI->connect...” line) has the correct database name, database user, and database password.
- ▶ Make sure the SQL query (found on the “my \$sql = ...” line) has the correct arguments and, most importantly, queries the correct table name.

Usage

The usage command is:

```
./check_bg1_midplane.pl
```

Return value

The return value is either:

- ▶ Return 0 (OK state in Nagios) if all midplanes have a status of “A”
- ▶ Return 2 (CRITICAL state in Nagios) if any midplanes have a status that is not “A”

Code

Example A-3 is the code to check the BG/L midplane.

Example: A-3 Check BG/L midplane

```
#!/usr/bin/perl
#####
###
### check_bg1_midplane.pl
### Created 6/25/2007
###
### Checks the Blue Gene DB2 database. Queries table TBGLMIDPLANE for any entries
### with status not equal to 'A'. It will return a critical status (exit status 2)
### and give a message of what Midplanes have errored. It will also write to a log
### file, located at /var/log/bgmr/nrpe/check_bg_midplane/midplane.log. If the
### problem is resolved, and a Midplane's status changes back to 'A', then the
### script will return OK ### status (exit status 0) and the log file will clear
### itself.
###
### This script is meant to be run by Nagios. Sysadmin alerting and notification
### is taken care of via Nagios. This script only checks the database and reports
### its findings.
```

```

###
#####
#Nagios Master Server
my $nagios_ip = "IP ADDRESS HERE";
#This Host's Linux Hostname
my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#Location of Web-viewable log file on the Nagios Master Server
my $master_log_loc = "WEB DIRECTORY HERE";
#Location of local log file on remote host
my $local_log_loc = "LOG DIRECTORY HERE";
#Name of the Log File
my $logfile = "midplane.log";
#HTML Link to add to the end of all errored output
my $loglink = "<br><a href='WEB URL HERE/midplane.html'>Click Here</a> to
view the log file.";
use DBI;
use DBD::DB2::Constants;
use DBD::DB2;
use strict;
#For Nagios
my $TIMEOUT = 15;
my
%ERRORS=('OK'=>0,'WARNING'=>1,'CRITICAL'=>2,'UNKNOWN'=>3,'DEPENDENT'=>4);
#
#DB2 Sourcing
#
my $db2result;
#
$db2result = `./discovery/db.src`;
#DB2 Connection String
my $dbh = DBI->connect("dbi:DB2:DB_NAME", "DB_USER", "DB_PASSWORD");
#If DB2 connection fails, print error and send CRITICAL ERROR
if(!(defined $dbh))
{
    print "CRITICAL : Error Connecting to DB2 Database:
$DBI::errstr\n";
    exit $ERRORS{'CRITICAL'};
}
#SQL Query
my $sql = "select serialnumber, machineserialnumber, posinmachine, status,
location, ismaster from tbglmidplane where status <> 'A' and status <> 'M' order
by serialnumber asc";
my $sth = $dbh->prepare($sql);
$sth->execute();
my ($serialnumber, $machineserialnumber, $posinmachine, $status,
$location, $ismaster);
my $count = 0;
my ($output, $shortoutput);
my ($serialleft, $serialright);
#Fetch DB2 data row by row
while ( ($serialnumber, $machineserialnumber, $posinmachine, $status,
$location, $ismaster) = $sth->fetchrow() )
{
    $count = $count + 1;
    #Remove Leading & Trailing White Space
    $serialnumber =~ s/^\s+//;
    $serialnumber =~ s/\s+$//;

```

```

$machineserialnumber =~ s/^\s+//;
$machineserialnumber =~ s/\s+$//;
$posinmachine =~ s/^\s+//;
$posinmachine =~ s/\s+$//;
$status =~ s/^\s+//;
$status =~ s/\s+$//;
$location =~ s/^\s+//;
$location =~ s/\s+$//;
$ismaster =~ s/^\s+//;
$ismaster =~ s/\s+$//;
$output .= "Midplane failed: Pos=$posinmachine Status=$status
Location=$location IsMaster=$ismaster <br><br> ";
}
$sth->finish();
$dbh->disconnect();
#If there are errors to print, write them to the log file.
open(MYOUTFILE, ">$local_log_loc/$logfile");
my $fileoutput = $output;
$fileoutput =~ s/<br><br>/\n/g;
print MYOUTFILE $fileoutput;
close(MYOUTFILE);
#SCP the Log File to the correct place on the Nagios Master Server,
#and set the permissions correctly.

#YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is
#to send a copy of the log file to the Nagios master server, put it in a
#folder under the Apache Web Directory, then change the permissions
#and the owner to be the nagios user and nagios group.
system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@${nagios_ip}:~USERID' >
/dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo mkdir -p $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc/$logfile \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null");
#If there are errors, print them and send a CRITICAL status.
if ($output ne "")
{
    #If full output is longer than Nagios allows,
    #Use the short output.
    if( length($output) + length($loglink) > 320 )
    {
        $output = "$count Midplanes Failed.$loglink";
    }
    else
    {
        $output = substr($output, 0, - 9);
    }
}

```

```

        $output .= $loglink;
    }
    print( $output, "\n" );
    exit $ERRORS{"CRITICAL"};
}
else
{
    print "OK : No Midplanes Failed\n";
    exit $ERRORS{"OK"};
}

```

A.4 Check BG/L node card

This script checks the Blue Gene DB2 database. It queries table TBGLNODECARD for any entries with status not equal to "A". It will return a critical status (exit status 2) and give a message indicating which node cards experienced errors. It will also write to a log file, which is located at /var/log/bgmr/nrpe/check_bg_linkcard/nodecard.log. If the problem is resolved and a node card's status changes back to "A", the script will return OK status (exit status 0) and the log file will clear itself.

Optional additions

You can run the:

- ▶ Check BG Node Event Handler script
- ▶ Build Log HTML script

This script is meant to be run by Nagios. The sysadmin alerting and notification script is taken care of via Nagios. This script only checks the database and reports its findings.

There are several variables at the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

nagios_ip	The IP address of the Nagios master server
master_log_loc	The location of the directory on the master host where the log file will be stored (in order to view it using the Nagios Web interface)
local_log_loc	The location of the directory on the remote host where the log file is stored
logfile	The name of the log file
loglink	The HTML link (and text) to the log file that will be viewed using the Nagios Web interface. This will be added to the end of all error-state output.

Other things to check

Check to:

- ▶ Make sure the DB2 Connection string (found on the "my \$dbh = DBI->connect..." line) has the correct database name, database user, and database password.
- ▶ Make sure the SQL query (found on the "my \$sql = ..." line) has the correct arguments and, most importantly, queries the correct table name.

Usage

The usage command is:

```
./check_bg1_nodocard.pl
```

Return value

The return value is either:

- ▶ Return 0 (OK state in Nagios) if all node cards have a status of "A"
- ▶ Return 2 (CRITICAL state in Nagios) if any node cards have a status that is not "A"

Code

Example A-4 is the code to check the BG/L node card.

Example: A-4 Check BG/L node card

```
#!/usr/bin/perl
#####
###
### check_bg1_nodocard.pl
### Created 6/25/2007
###
### Checks the Blue Gene DB2 database. Queries table
### TBGLNODECARD for any entries with status not
### equal to 'A'. It will return a critical status (exit status
### 2) and give a message of what Node Cards have errored. It
### will also write to a log file, located at
### /var/log/bgmr/nrpe/check_bg_nodocard/nodocard.log.
### If the problem is resolved, and a Node Card's status
### changes back to 'A', then the script will return OK
### status (exit status 0) and the log file will clear
### itself.
###
### This script is meant to be run by Nagios. Sysadmin
### alerting and notification is taken care of via Nagios.
### This script only checks the database and reports its
### findings.
###
#####
#Nagios Master Server
my $nagios_ip = "IP ADDRESS HERE";
#This Host's Linux Hostname
my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#Location of Web-viewable log file on the Nagios Master Server
my $master_log_loc = "WEB DIRECTORY HERE";
#Location of local log file on remote host
my $local_log_loc = "LOG DIRECTORY HERE";
#Name of the Log File
my $logfile = "nodocard.log";
#HTML Link to add to the end of all errored output
my $loglink = "<br><a href='WEB URL HERE/nodocard.html'>Click Here</a> to
view the log file.";

use DBI;
use DBD::DB2::Constants;
```

```

use DBD::DB2;
use strict;
#For Nagios
my $TIMEOUT = 15;
my
%ERRORS=('OK'=>0,'WARNING'=>1,'CRITICAL'=>2,'UNKNOWN'=>3,'DEPENDENT'=>4);

#      #DB2 Sourcing
#      my $db2result;
#      $db2result = `./discovery/db.src`;
#      #DB2 Connection String
my $dbh = DBI->connect("dbi:DB2:DB_NAME", "DB_USER", "DB_PASSWORD");
#If DB2 connection fails, print error and send CRITICAL ERROR
if(!(defined $dbh))
{
    print "CRITICAL : Error Connecting to DB2 Database:
$DBI::errstr\n";
    exit $ERRORS{'CRITICAL'};
}
#SQL Query
my $sql = "Select serialnumber, status, location, hasidochip from
tblnodecard where status <> 'A' and status <> 'M'";
my $sth = $dbh->prepare($sql);
$sth->execute();
my ($serialnumber, $status, $location, $hasido);
my $count = 0;
my ($output, $shortoutput);
my ($serialleft, $serialright);
#Fetch DB2 data row by row
while ( ($serialnumber, $status, $location, $hasido) = $sth->fetchrow() )
{
    $count = $count + 1;
    #Remove Leading & Trailing White Space
    $serialnumber =~ s/^\s+//;
    $serialnumber =~ s/\s+$//;
    $status =~ s/^\s+//;
    $status =~ s/\s+$//;
    $location =~ s/^\s+//;
    $location =~ s/\s+$//;
    $hasido =~ s/^\s+//;
    $hasido =~ s/\s+$//;
    $output .= "Node Card failed: Status=$status Location=$location
HasIDO=$hasido <br><br> ";
}
$sth->finish();
$dbh->disconnect();
#If there are errors to print, write them to the log file.
open(MYOUTFILE, ">$local_log_loc/$logfile");
my $fileoutput = $output;
$fileoutput =~ s/<br><br>/\n/g;
print MYOUTFILE $fileoutput;
close(MYOUTFILE);
#SCP the Log File to the correct place on the Nagios Master Server,
#and set the permissions correctly.

```



```

#YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is
#to send a copy of the log file to the Nagios master server, put it in a
#folder under the Apache Web Directory, then change the permissions
#and the owner to be the nagios user and nagios group.
system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@${nagios_ip}:~USERID' >
/dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo mkdir -p $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc/$logfile \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null");
#If there are errors, print them and send a CRITICAL status.
if ($output ne "")
{
    #If full output is longer than Nagios allows,
    #Use the short output.
    if( length($output) + length($loglink) > 320 )
    {
        $output = "$count Node Cards Failed.$loglink";
    }
    else
    {
        $output = substr($output, 0, - 9);
        $output .= $loglink;
    }
    print( $output, "\n" );
    exit $ERRORS{"CRITICAL"};
}
else
{
    print "OK : No Node Cards Failed\n";
    exit $ERRORS{"OK"};
}

```

A.5 Check BG/L node

This script checks the Blue Gene DB2 database. It queries table TBGLNODE for any entries with status not equal to "A". It will return a critical status (exit status 2) and give a message indicating which nodes experienced errors. It will also write to a log file, which is located at /var/log/bgmr/nrpe/check_bg_node/node.log. If the problem is resolved and a node's status changes back to "A", the script will return OK status (exit status 0) and the log file will clear itself.

Optional additions

You can run the:

- ▶ Check BG Node Event Handler script
- ▶ Build Log HTML script

This script is meant to be run by Nagios. The sysadmin alerting and notification script is taken care of via Nagios. This script only checks the database and reports its findings.

There are several variables at the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

nagios_ip	The IP address of the Nagios master server
master_log_loc	The location of the directory on the master host where the log file will be stored (in order to view it using the Nagios Web interface)
local_log_loc	The location of the directory on the remote host where the log file is stored
logfile	The name of the log file
loglink	The HTML link (and text) to the log file that will be viewed using the Nagios Web interface. This will be added to the end of all error-state output.

Other things to check

Check to:

- ▶ Make sure the DB2 Connection string (found on the “my \$dbh = DBI->connect...” line) has the correct database name, database user, and database password.
- ▶ Make sure the SQL query (found on the “my \$sql = ...” line) has the correct arguments and, most importantly, queries the correct table name.

Usage

The usage command is:

```
./check_bg1_node.pl
```

Return value

The return value is either:

- ▶ Return 0 (OK State in Nagios) if all nodes have a status of “A”
- ▶ Return 2 (CRITICAL State in Nagios) if any nodes have a status that is not “A”

Code

Example A-5 is the code to check the BG/L node.

Example: A-5 Check BG/L node

```
#!/usr/bin/perl
#####
###
### check_bg1_node.pl
### Created 6/25/2007
###
### Checks the Blue Gene DB2 database. Queries table
### TBGLNODE for any entries with status not
### equal to 'A'. It will return a critical status (exit status
```

```

### 2) and give a message of what Nodes have errored. It
### will also write to a log file, located at
### /var/log/bgmr/nrpe/check_bg_node/node.log.
### If the problem is resolved, and a Node's status
### changes back to 'A', then the script will return OK
### status (exit status 0) and the log file will clear
### itself.
###
### This script is meant to be run by Nagios. Sysadmin
### alerting and notification is taken care of via Nagios.
### This script only checks the database and reports its
### findings.
###
#####
#Nagios Master Server
my $nagios_ip = "IP ADDRESS HERE";
#This Host's Linux Hostname
my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#Location of Web-viewable log file on the Nagios Master Server
my $master_log_loc = "WEB DIRECTORY HERE";
#Location of local log file on remote host
my $local_log_loc = "LOG DIRECTORY HERE";
#Name of the Log File
my $logfile = "node.log";
#HTML Link to add to the end of all errored output
my $loglink = "<br><a href='WEB URL HERE/node.html'>Click Here</a> to view
the log file.";

use DBI;
use DBD::DB2::Constants;
use DBD::DB2;
use strict;
#For Nagios
my $TIMEOUT = 15;
my
%ERRORS=('OK'=>0,'WARNING'=>1,'CRITICAL'=>2,'UNKNOWN'=>3,'DEPENDENT'=>4);
#
#DB2 Sourcing
#
my $db2result;
#
$db2result = `./discovery/db.src`;
#DB2 Connection String
my $dbh = DBI->connect("dbi:DB2:DB_NAME", "DB_USER", "DB_PASSWORD");
#If DB2 connection fails, print error and send CRITICAL ERROR
if(!(defined $dbh))
{
    print "CRITICAL : Error Connecting to DB2 Database:
$DBI::errstr\n";
    exit $ERRORS{'CRITICAL'};
}
#SQL Query
my $sql = "Select serialnumber, status, location from tbglnode where
status <> 'A' and status <> 'M'";
my $sth = $dbh->prepare($sql);
$sth->execute();
my ($serialnumber, $status, $location);

```

```

my $count = 0;
my ($output, $shortoutput);
my ($serialleft, $serialright);
#Fetch DB2 data row by row
while ( ($serialnumber, $status, $location) = $sth->fetchrow() )
{
    $count = $count + 1;
    #Remove Leading & Trailing White Space
    $serialnumber =~ s/^\s+//;
    $serialnumber =~ s/\s+$//;
    $status =~ s/^\s+//;
    $status =~ s/\s+$//;
    $location =~ s/^\s+//;
    $location =~ s/\s+$//;
    $output .= "Node failed: Status=$status Location=$location
<br><br> ";
}
$sth->finish();
$dbh->disconnect();
#If there are errors to print, write them to the log file.
open(MYOUTFILE, ">$local_log_loc/$logfile");
my $fileoutput = $output;
$fileoutput =~ s/<br><br>/\n/g;
print MYOUTFILE $fileoutput;
close(MYOUTFILE);
#SCP the Log File to the correct place on the Nagios Master Server,
#and set the permissions correctly.

#YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is
#to send a copy of the log file to the Nagios master server, put it in a
#folder under the Apache Web Directory, then change the permissions
#and the owner to be the nagios user and nagios group.
system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@${nagios_ip}:~USERID' >
/dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo mkdir -p $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc/$logfile \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null");
#If there are errors, print them and send a CRITICAL status.
if ($output ne "")
{
    #If full output is longer than Nagios allows,
    #Use the short output.
    if( length($output) + length($loglink) > 320 )
    {
        $output = "$count Nodes Failed.$loglink";
    }
}

```

```

    }
    else
    {
        $output = substr($output, 0, - 9);
        $output .= $loglink;
    }
    print( $output, "\n" );
    exit $ERRORS{"CRITICAL"};
}
else
{
    print "OK : No Nodes Failed\n";
    exit $ERRORS{"OK"};
}

```

A.6 Check BG/L processor card

This script checks the Blue Gene DB2 database. It queries table TBGLPROCESSORCARD for any entries with status not equal to "A". It will return a critical status (exit status 2) and give a message of which processor cards experienced errors. It will also write to a log file, which is located at /var/log/bgmr/nrpe/check_bg_processorcard/processorcard.log. If the problem is resolved and a processor card's status changes back to "A", the script will return OK status (exit status 0) and the log file will clear itself.

Optional additions

You can also run the:

- Check BG Processor Card Event Handler script
- Build Log HTML script

This script is meant to be run by Nagios. The sysadmin alerting and notification script is taken care of via Nagios. This script only checks the database and reports its findings.

There are several variables at the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

nagios_ip	The IP address of the Nagios master server
master_log_loc	The location of the directory on the master host where the log file will be stored (in order to view it using the Nagios Web interface)
local_log_loc	The location of the directory on the remote host where the log file is stored
logfile	The name of the log file
loglink	The HTML link (and text) to the log file that will be viewed using the Nagios Web interface. This will be added to the end of all error-state output

Other things to check

Be sure that you:

- Make sure the DB2 Connection string (found on the "my \$dbh = DBI->connect..." line) has the correct database name, database user, and database password.

- Make sure the SQL query (found on the “my \$sql = ...” line) has the correct arguments and, most importantly, queries the correct table name.

Usage

The usage command is:

```
./check_bg1_processorcard.pl
```

Return value

The return value is either:

- Return 0 (OK State in Nagios) if all processor cards have a status of “A”
- Return 2 (CRITICAL State in Nagios) if any processor cards have a status that is not “A”

Code

Example A-6 is the code to check the BG/L processor card.

Example: A-6 Check BG/L processor card

```
#!/usr/bin/perl
#####
###
### check_bg1_processorcard.pl
### Created 6/25/2007
###
### Checks the Blue Gene DB2 database. Queries table
### TBGLPROCESSORCARD for any entries with status not
### equal to 'A'. It will return a critical status (exit status
### 2) and give a message of what Processor Cards have errored. It
### will also write to a log file, located at
### /var/log/bgmr/nrpe/check_bg_processorcard/processorcard.log.
### If the problem is resolved, and a Processor Card's status
### changes back to 'A', then the script will return OK
### status (exit status 0) and the log file will clear
### itself.
###
### This script is meant to be run by Nagios. Sysadmin
### alerting and notification is taken care of via Nagios.
### This script only checks the database and reports its
### findings.
###
#####
#Nagios Master Server
my $nagios_ip = "IP ADDRESS HERE";
#This Host's Linux Hostname
my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#Location of Web-viewable log file on the Nagios Master Server
my $master_log_loc = "WEB DIRECTORY HERE";
#Location of local log file on remote host
my $local_log_loc = "LOG DIRECTORY HERE";
#Name of the Log File
my $logfile = "processorcard.log";
#HTML Link to add to the end of all errored output
```

```

my $loglink = "<br><a href='WEB URL HERE/processorcard.html'>Click
Here</a> to view the log file.";
use DBI;
use DBD::DB2::Constants;
use DBD::DB2;
use strict;
#For Nagios
my $TIMEOUT = 15;
my
%ERRORS=('OK'=>0,'WARNING'=>1,'CRITICAL'=>2,'UNKNOWN'=>3,'DEPENDENT'=>4);
#      #DB2 Sourcing
#      my $db2result;
#      $db2result = `./discovery/db.src`;
#      #DB2 Connection String
my $dbh = DBI->connect("dbi:DB2:DB_NAME", "DB_USER", "DB_PASSWORD");
#If DB2 connection fails, print error and send CRITICAL ERROR
if(!(defined $dbh))
{
    print "CRITICAL : Error Connecting to DB2 Database:
$DBI::errstr\n";
    exit $ERRORS{'CRITICAL'};
}
#SQL Query
my $sql = "Select serialnumber, status, location from tbglprocessorcard
where status <> 'A' and status <> 'M'";
my $sth = $dbh->prepare($sql);
$sth->execute();
my ($serialnumber, $status, $location);
my $count = 0;
my ($output, $shortoutput);
my ($serialleft, $serialright);
#Fetch DB2 data row by row
while ( ($serialnumber, $status, $location) = $sth->fetchrow() )
{
    $count = $count + 1;
    #Remove Leading & Trailing White Space
    $serialnumber =~ s/^\s+//;
    $serialnumber =~ s/\s+$//;
    $status =~ s/^\s+//;
    $status =~ s/\s+$//;
    $location =~ s/^\s+//;
    $location =~ s/\s+$//;
    $output .= "Processor Card failed: Status=$status
Location=$location <br><br> ";
}
$sth->finish();
$dbh->disconnect();
#If there are errors to print, write them to the log file.
open(MYOUTFILE, ">$local_log_loc/$logfile");
my $fileoutput = $output;
$fileoutput =~ s/<br><br>/\n/g;
print MYOUTFILE $fileoutput;
close(MYOUTFILE);
#SCP the Log File to the correct place on the Nagios Master Server,
#and set the permissions correctly.

```

```

#YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is
#to send a copy of the log file to the Nagios master server, put it in a
#folder under the Apache Web Directory, then change the permissions
#and the owner to be the nagios user and nagios group.
system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@${nagios_ip}:~USERID' >
/dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo mkdir -p $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc/$logfile \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null");
#If there are errors, print them and send a CRITICAL status.
if ($output ne "")
{
    #If full output is longer than Nagios allows,
    #Use the short output.
    if( length($output) + length($loglink) > 320 )
    {
        $output = "$count Processor Cards Failed.$loglink";
    }
    else
    {
        $output = substr($output, 0, - 9);
        $output .= $loglink;
    }
    print( $output, "\n" );
    exit $ERRORS{"CRITICAL"};
}
else
{
    print "OK : No Processor Cards Failed\n";
    exit $ERRORS{"OK"};
}

```

A.7 Check BG/L service action

Checks the Blue Gene DB2 database. This script queries table TBGLSERVICEACTION for any entries with status not equal to “C”. It will return a critical status (exit status 2) and give a message indicating which service actions are unresolved. It will also write to a log file, which is located at /var/log/bgmr/nrpe/check_bg_serviceaction/serviceaction.log. If the problem is resolved and a service action’s status changes back to “C”, the script will return OK status (exit status 0) and the log file will clear itself.

Optional additions

You can also run the:

- ▶ Check BG Service Action Event Handler script
- ▶ Build Log HTML script

This script is meant to be run by Nagios. The sysadmin alerting and notification script is taken care of via Nagios. This script only checks the database and reports its findings.

There are several variables at the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

nagios_ip	The IP address of the Nagios master server
master_log_loc	The location of the directory on the master host where the log file will be stored (in order to view it using the Nagios Web interface)
local_log_loc	The location of the directory on the remote host where the log file is stored
logfile	The name of the log file
loglink	The HTML link (and text) to the log file that will be viewed using the Nagios Web interface. This will be added to the end of all error-state output.

Other things to check

Also check that you:

- ▶ Make sure the DB2 Connection string (found on the “my \$dbh = DBI->connect...” line) has the correct database name, database user, and database password.
- ▶ Make sure the SQL query (found on the “my \$sql = ...” line) has the correct arguments and, most importantly, queries the correct table name.

Usage

The command to check usage is:

```
./check_bg1_serviceaction.pl
```

Return value

The return value is either:

- ▶ Return 0 (OK State in Nagios) if all service actions have a status of “A”
- ▶ Return 2 (CRITICAL State in Nagios) if any service actions have a status that is not “A”

Code

Example A-7 is the code to check the BG/L service action.

Example: A-7 Check BG/L service action

```
#!/usr/bin/perl
#####
###
### check_bg1_serviceaction.pl
### Created 6/25/2007
###
### Checks the Blue Gene DB2 database. Queries table
### TBGLSERVICEACTION for any entries with status not
### equal to 'C'. It will return a critical status (exit status
```

```

### 2) and give a message of what Service Actions are unresolved.
### It will also write to a log file, located at
### /var/log/bgmr/nrpe/check_bg_serviceaction/serviceaction.log.
### If the problem is resolved, and a Service Action's status
### changes back to 'C', then the script will return OK
### status (exit status 0) and the log file will clear
### itself.
###
### This script is meant to be run by Nagios. Sysadmin
### alerting and notification is taken care of via Nagios.
### This script only checks the database and reports its
### findings.
###
#####
#Nagios Master Server
my $nagios_ip = "IP ADDRESS HERE";
#This Host's Linux Hostname
my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#Location of Web-viewable log file on the Nagios Master Server
my $master_log_loc = "WEB DIRECTORY HERE";
#Location of local log file on remote host
my $local_log_loc = "LOG DIRECTORY HERE";
#Name of the Log File
my $logfile = "serviceaction.log";
#HTML Link to add to the end of all errored output
my $loglink = "<br><a href='WEB URL HERE/serviceaction.html'>Click
Here</a> to view the log file.";
use DBI;
use DBD::DB2::Constants;
use DBD::DB2;
use strict;
#For Nagios
my $TIMEOUT = 15;
my
%ERRORS=('OK'=>0,'WARNING'=>1,'CRITICAL'=>2,'UNKNOWN'=>3,'DEPENDENT'=>4);
# #DB2 Sourcing
# my $db2result;
# $db2result = `./discovery/db.src`;
#DB2 Connection String
my $dbh = DBI->connect("dbi:DB2:DB_NAME", "DB_USER", "DB_PASSWORD");
#If DB2 connection fails, print error and send CRITICAL ERROR
if(!(defined $dbh))
{
    print "CRITICAL : Error Connecting to DB2 Database:
$DBI::errstr\n";
    exit $ERRORS{'CRITICAL'};
}
#SQL Query
my $sql = "select id, location, status, usernameprepareforservice from
tblserviceaction where status <> 'C' order by id asc";
my $sth = $dbh->prepare($sql);
$sth->execute();
my ($id, $location, $status, $usernameprepareforservice);
my $count = 0;

```

```

my ($output, $shortoutput);
#Fetch DB2 data row by row
while ( ($id, $location, $status, $usernameprepareforservice) =
$sth->fetchrow() )
{
    $count = $count + 1;
    #Remove Leading & Trailing White Space
    $id =~ s/^\s+//;
    $id =~ s/\s+$//;
    $location =~ s/^\s+//;
    $location =~ s/\s+$//;
    $usernameprepareforservice =~ s/^\s+//;
    $usernameprepareforservice =~ s/\s+$//;
    $status =~ s/^\s+//;
    $status =~ s/\s+$//;
    $output .= "Service Action $id unresolved: Status=$status
Location=$location Username=$usernameprepareforservice <br><br> ";
}
$sth->finish();
$dbh->disconnect();
#If there are errors to print, write them to the log file.
open(MYOUTFILE, ">$local_log_loc/$logfile");
my $fileoutput = $output;
$fileoutput =~ s/<br><br>/\n/g;
print MYOUTFILE $fileoutput;
close(MYOUTFILE);
#SCP the Log File to the correct place on the Nagios Master Server,
#and set the permissions correctly.

#YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is
#to send a copy of the log file to the Nagios master server, put it in a
#folder under the Apache Web Directory, then change the permissions
#and the owner to be the nagios user and nagios group.
system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@${nagios_ip}:~USERID' >
/dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo mkdir -p $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc/$logfile \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null");
#If there are errors, print them and send a CRITICAL status.
if ($output ne "")
{
    #If full output is longer than Nagios allows,
    #Use the short output.
    if( length($output) + length($loglink) > 320 )
    {

```

```

        $output = "$count Service Actions Unresolved.$loglink";
    }
    else
    {
        $output = substr($output, 0, - 9);
        $output .= $loglink;
    }
    print( $output, "\n" );
    exit $ERRORS{"CRITICAL"};
}
else
{
    print "OK : No Service Actions Unresolved\n";
    exit $ERRORS{"OK"};
}

```

A.8 Check BG/L service card

This script checks the Blue Gene DB2 database. It queries table TBGLSERVICECARD for any entries with status not equal to "A". It will return a critical status (exit status 2) and give a message indicating which service cards experienced errors. It will also write to a log file, which is located at `/var/log/bgmr/nrpe/check_bg_servicecard/servicecard.log`. If the problem is resolved and a service card's status changes back to "A", the script will return OK status (exit status 0) and the log file will clear itself.

Optional additions

You can also run the:

- ▶ Check BG Service Card Event Handler script
- ▶ Build Log HTML script

This script is meant to be run by Nagios. The sysadmin alerting and notification script is taken care of via Nagios. This script only checks the database and reports its findings.

There are several variables at the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

nagios_ip	The IP address of the Nagios master server
master_log_loc	The location of the directory on the master host where the log file will be stored (in order to view it using the Nagios Web interface)
local_log_loc	The location of the directory on the remote host where the log file is stored
logfile	The name of the log file
loglink	The HTML link (and text) to the log file that will be viewed using the Nagios Web interface. This will be added to the end of all error-state output

Other things to check

Check to:

- ▶ Make sure the DB2 Connection string (found on the "my \$dbh = DBI->connect..." line) has the correct database name, database user, and database password.

- Make sure the SQL query (found on the “my \$sql = ...” line) has the correct arguments and, most importantly, queries the correct table name.

Usage

The usage command is:

```
./check_bg1_servicecard.pl
```

Return value

The return value is either:

- Return 0 (OK State in Nagios) if all service card have a status of “A”
- Return 2 (CRITICAL State in Nagios) if any service cards have a status that is not “A”

Code

Example A-8 is the code to check the BG/L service card.

Example: A-8 Check BG/L service card

```
#!/usr/bin/perl
#####
###
### check_bg1_servicecard.pl
### Created 6/25/2007
###
### Checks the Blue Gene DB2 database. Queries table
### TBGLSERVICECARD for any entries with status not
### equal to 'A'. It will return a critical status (exit status
### 2) and give a message of what Service Cards have errored. It
### will also write to a log file, located at
### /var/log/bgmr/nrpe/check_bg_servicecard/servicecard.log.
### If the problem is resolved, and a Service Card's status
### changes back to 'A', then the script will return OK
### status (exit status 0) and the log file will clear
### itself.
###
### This script is meant to be run by Nagios. Sysadmin
### alerting and notification is taken care of via Nagios.
### This script only checks the database and reports its
### findings.
###
#####
#Nagios Master Server
my $nagios_ip = "IP ADDRESS HERE";
#This Host's Linux Hostname
my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#Location of Web-viewable log file on the Nagios Master Server
my $master_log_loc = "WEB DIRECTORY HERE";
#Location of local log file on remote host
my $local_log_loc = "LOG DIRECTORY HERE";
#Name of the Log File
my $logfile = "serviceaction.log";
#HTML Link to add to the end of all errored output
```

```

my $loglink = "<br><a href='WEB URL HERE/serviceaction.html'>Click
Here</a> to view the log file.";

```

```

use DBI;
use DBD::DB2::Constants;
use DBD::DB2;
use strict;
#For Nagios
my $TIMEOUT = 15;
my
%ERRORS=('OK'=>0,'WARNING'=>1,'CRITICAL'=>2,'UNKNOWN'=>3,'DEPENDENT'=>4);
#      #DB2 Sourcing
#      my $db2result;
#      $db2result = `./discovery/db.src`;
#DB2 Connection String
my $dbh = DBI->connect("dbi:DB2:DB_NAME", "DB_USER", "DB_PASSWORD");
#If DB2 connection fails, print error and send CRITICAL ERROR
if(!(defined $dbh))
{
    print "CRITICAL : Error Connecting to DB2 Database:
$DBI::errstr\n";
    exit $ERRORS{'CRITICAL'};
}
#SQL Query
my $sql = "select serialnumber, midplaneserialnumber, posinmidplane,
status, location from tbglservicecard where status <> 'A' and status <> 'M' order
by serialnumber asc";
my $sth = $dbh->prepare($sql);
$sth->execute();
my ($serialnumber, $midplaneserialnumber, $posinmidplane, $status,
$location);
my $count = 0;
my ($output, $shortoutput);
my ($serialleft, $serialright);
#Fetch DB2 data row by row
while ( ($serialnumber, $midplaneserialnumber, $posinmidplane, $status,
$location) = $sth->fetchrow() )
{
    $count = $count + 1;
    #Remove Leading & Trailing White Space
    $serialnumber =~ s/^\s+//;
    $serialnumber =~ s/\s+$//;
    $midplaneserialnumber =~ s/^\s+//;
    $midplaneserialnumber =~ s/\s+$//;
    $posinmidplane =~ s/^\s+//;
    $posinmidplane =~ s/\s+$//;
    $status =~ s/^\s+//;
    $status =~ s/\s+$//;
    $location =~ s/^\s+//;
    $location =~ s/\s+$//;
    $output .= "Service Card failed: Pos=$posinmidplane Status=$status
Location=$location <br><br> ";
}
$sth->finish();
$dbh->disconnect();

```

```

#If there are errors to print, write them to the log file.
open(MYOUTFILE, ">$local_log_loc/$logfile");
    my $fileoutput = $output;
    $fileoutput =~ s/<br><br>/\n/g;
    print MYOUTFILE $fileoutput;
close(MYOUTFILE);
#SCP the Log File to the correct place on the Nagios Master Server,
#and set the permissions correctly.

#YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is
#to send a copy of the log file to the Nagios master server, put it in a
#folder under the Apache Web Directory, then change the permissions
#and the owner to be the nagios user and nagios group.
system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@${nagios_ip}:~USERID' >
/dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo mkdir -p $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc/$logfile \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null");
    #If there are errors, print them and send a CRITICAL status.
    if ($output ne "")
    {
        #If full output is longer than Nagios allows,
        #Use the short output.
        if( length($output) + length($loglink) > 320 )
        {
            $output = "$count Service Cards Failed. $loglink";
        }
        else
        {
            $output = substr($output, 0, - 9);
            $output .= $loglink;
        }
        print( $output, "\n" );
        exit $ERRORS{"CRITICAL"};
    }
    else
    {
        print "OK : No Service Cards Failed\n";
        exit $ERRORS{"OK"};
    }
}

```

Archived

Blue Gene/P plug-ins

These are plug-in scripts for running Nagios on a Blue Gene (BG) system.

Soft copies of these scripts are available online in zipped TGZ format at:

<ftp://www.redbooks.ibm.com/redbooks/REDP4356>

B.1 Check BG/P jobs

This script checks the Blue Gene DB2 database. It queries table TBGPJOB_HISTORY for any entries with exit status not equal to 0. It will return a warning status (exit status 1) and give a message indicating which job IDs experienced errors. It will also write to a log file, which is located at `/var/log/bgmr/nagios/check_bg_jobs/joberror.log`. In order to not continue returning the same job with an error, two files will keep a record of the last job with an error found (`/var/lib/bgmr/nagios/check_bg_jobs/jobid.error`) and the last job with an error cleared by the system administrators via Nagios (`/var/lib/bgmr/nagios/check_bg_jobs/jobid.clear`).

Additional requirements

You also must run the:

- ▶ Check BG Jobs Event Handler script
- ▶ Build Log HTML script

When run in Nagios, Check BG Jobs will never return to an “OK” state from a “WARNING” state automatically. The user or system administrator must manually submit a “Passive Service Check Result” via Nagios. You can do this in one of two ways. This is where the two “Additional Requirements” scripts just listed come into play; make sure they are properly installed and configured in Nagios before proceeding.

When a job error is presented in Nagios, the status information section will link to a log file. At the top of this page, there is a text box and a submit button. When you enter the text that you want to locate in the Status Information section and click Submit to clear the log file, this action submits an OK status to Nagios, and not receive further warnings about any jobs that failed before this point. If Check BG Jobs reports an error again, it will be because a job has failed since the last time you submitted this OK state. Basically, use the text box and button to clear any errors you have disregarded or already handled.

This script is meant to be run by Nagios. The sysadmin alerting and notification script is taken care of via Nagios. This script only checks the database and reports its findings.

There are several variables at the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

nagios_ip	The IP address of the Nagios master server
master_log_loc	The location of the directory on the master host where the log file will be stored (in order to view it using the Nagios Web interface)
local_log_loc	The location of the directory on the remote host where the log file is stored
logfile	The name of the log file
loglink	The HTML link (and text) to the log file that will be viewed using the Nagios Web interface. This will be added to the end of all error-state output

Other things to check

Check to:

- ▶ Make sure the paths to `jobid.error` and `jobid.clear` are correct.
- ▶ Make sure the DB2 Connection string (found on the “my \$dbh = DBI->connect...” line) has the correct database name, database user, and database password.
- ▶ Make sure the SQL query (found on the “my \$sql = ...” line) has the correct arguments and, most importantly, queries the correct table name.

Usage

The check usage command is:

```
./check_bgp_jobs.pl
```

Return value

The return value is either:

- ▶ Return 0 (OK State in Nagios) if no new failed jobs are found since the last “OK” by the user.
- ▶ Return 1 (WARNING State in Nagios) if one or more new failed jobs are found since the last “OK” by the user.

Code

Example B-1 is the code to check the BG/P jobs.

Example: B-1 Check BG/P jobs

```
#!/usr/bin/perl
#####
###
### check_bgp_jobs.pl
### Created 6/20/2007
###
### Checks the Blue Gene DB2 database. Queries table TBGPJOB_HISTORY for any
### entries with exit status not equal to 0. It will return a warning status (exit
### status 1) and give a message of what job IDs have errored. It will also write
### to a log file, located at /var/log/bgmr/nagios/check_bg_jobs/joberror.log. In
### order to not keep returning the same errored job, two files will keep a record
### of the last errored job found (/var/lib/bgmr/nagios/check_bg_jobs/jobid.err)
### and the last errored job cleared by the system administrators via Nagios
### (/var/lib/bgmr/nagios/check_bg_jobs/jobid.clean).
###
### This script is meant to be run by Nagios. Sysadmin alerting and notification
### is taken care of via Nagios. This script only checks the database and reports
### its findings.
#####
#Nagios Master Server
my $nagios_ip = "IP ADDRESS HERE";
#This Host's Linux Hostname
my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#Location of Web-viewable log file on the Nagios Master Server
my $master_log_loc = "WEB DIRECTORY HERE";
#Location of local log file on remote host
my $local_log_loc = "LOG DIRECTORY HERE";
#Name of the Log File
my $logfile = "joberrors.log";
#HTML Link to add to the end of all errored output
my $loglink = "<br><a href='WEB URL HERE/joberrors.html'>Click Here</a> to
view the log file.";

use DBI;
use DBD::DB2::Constants;
use DBD::DB2;
use strict;
```

```

# Nagios specific
my $TIMEOUT = 15;
my
%ERRORS=('OK'=>0,'WARNING'=>1,'CRITICAL'=>2,'UNKNOWN'=>3,'DEPENDENT'=>4);
#
# DB2 Sourcing
#
my $db2result;
#
$db2result = `./usr/local/bin/db.src`;
#DB2 Connection String
my $dbh = DBI->connect("dbi:DB2:DB_NAME", "DB_USER", "DB_PASSWORD");
#If DB2 connection fails, print error and send CRITICAL ERROR
if(!(defined $dbh))
{
    print "DB2 Error: $DBI::errstr\n";
    exit 2;
}
#Read the JobID of the last cleared job.
open(CLEANIDINFILE, "<TATE_FILE_DIRECTORY_HERE/jobid.clear");
my @file = <CLEANIDINFILE>;
my $cleanmaxid = $file[0];
chomp($cleanmaxid);
close(CLEANIDINFILE);
#Read the JobID of the last job found.
open(ERRIDINFILE, "<STATE_FILE_DIRECTORY_HERE/jobid.error");
my @file = <ERRIDINFILE>;
my $errmaxid = $file[0];
chomp($errmaxid);
close(ERRIDINFILE);
#SQL Query
my $sql = "select jobid, username, jobname, blockid, exitstatus, errtext,
executable, args, char(entrydate) from tbgpjob_history where exitstatus<>0 and
jobid > $cleanmaxid order by jobid asc";
my $sth = $dbh->prepare($sql);
$sth->execute();
my( $jobid, $username, $jobname, $blockid, $exitstatus, $errtext,
$executable, $args, $entrydate);
my ($output, $fileoutput);
my $date;
my $time;
my $shortoutput;
my $count = 0;
my $maxid = 0;
#Fetch DB2 data row by row
while( ( $jobid, $username, $jobname, $blockid, $exitstatus, $errtext,
$executable, $args, $entrydate) = $sth->fetchrow() )
{
    $count = $count + 1;
    #Remove Leading & Trailing White Space
    $jobid =~ s/^\s+//;
    $jobid =~ s/\s+$//;
    $username =~ s/^\s+//;
    $username =~ s/\s+$//;
    $jobname =~ s/^\s+//;
    $jobname =~ s/\s+$//;
    $exitstatus =~ s/^\s+//;
    $exitstatus =~ s/\s+$//;

```

```

$blockid =~ s/^\s+//;
$blockid =~ s/\s+$//;
$errtext =~ s/^\s+//;
$errtext =~ s/\s+$//;
$executable =~ s/^\s+//;
$executable =~ s/\s+$//;
$args =~ s/^\s+//;
$args =~ s/\s+$//;
$entrydate =~ s/^\s+//;
$entrydate =~ s/\s+$//;
if( $jobid !~ m/^\$/ )
{
    $shortoutput .= "$jobid, ";
}
#Split EntryDate into Date and Time
$entrydate =~ m/(\d+-\d+-\d+)-(\d+\.\d+\.\d+)/;
$date = $1;
$time = $2;
$output .= "JOB $jobid FAILED $date $time: User: $username, Name:
$jobname, Exit: $exitstatus, Error: $errtext, Block: $blockid Exe (with args):
${executable} ${args} <br><br> ";

#If JobID is greater than the last JobID found.
if($jobid > $errmaxid)
{
    $fileoutput .= "JOB $jobid FAILED $date $time: User:
$username, Name: $jobname, Exit: $exitstatus, Error: $errtext, Block: $blockid \n
Exe (with args): ${executable} ${args}\n\n";
    $maxid = $jobid;
}
}

#Disconnect from DB2
$sth->finish();
$dbh->disconnect();
#Set a new max JobID, write it to the file.
if ($maxid != 0)
{
    $errmaxid = $maxid;
}
open(IDOUTFILE, ">STATE_FILE_DIRECTORY_HERE/jobid.error");
print IDOUTFILE $errmaxid;
close(IDOUTFILE);
#If there are errors to print, write them to the log file.
if( $fileoutput ne "")
{
    open(JOBLOGFILE, ">>$local_log_loc/$logfile");
    print JOBLOGFILE $fileoutput;
    close(JOBLOGFILE);
}
#SCP the Log File to the correct place on the Nagios Master Server, and
#set the permissions correctly.

#YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is to
#send a copy of the log file to the Nagios master server, put it in a

```

```

        #folder under the Apache Web Directory, then change the permissions
        #and the owner to be the nagios user and nagios group.
        system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@$${nagios_ip}:~USERID' >
/dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo mkdir -p $master_log_loc \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc/$logfile \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null");
        $shortoutput = substr($shortoutput, 0, - 2);
        #If length of short output is longer than Nagios allows,
        #Shorten it further.
        if( (length($shortoutput) + length("$count Jobs Failed: ") +
length($loglink)) > 230 )
        {
            $shortoutput = "$count Jobs Failed. $loglink";
        }
        else
        {
            $shortoutput = "$count Jobs Failed: $shortoutput. $loglink";
        }
        if( $count > 0 )
        {
            #If full output is longer than nagios allows,
            #Use the short output.
            if( length($output) + length($loglink) > 320 )
            {
                $output = $shortoutput;
            }
            else
            {
                $output = substr($output, 0, - 9);
                $output .= $loglink;
            }
            print( $output);
            exit $ERRORS{"WARNING"};
        }
        else
        {
            print "OK : No Failed Jobs\n";
            exit $ERRORS{"OK"};
        }
    }
}

```

B.2 Check BG/P link card

This script checks the Blue Gene DB2 database. It queries table TBGPLINKCARD for any entries with status not equal to "A". It will return a critical status (exit status 2) and give a message indicating which link cards experienced errors. It will also write to a log file, which is located at `var/log/bgmr/nrpe/check_bg_linkcard/linkcard.log`. If the problem is resolved and a link card's status changes back to "A", the script will return OK status (exit status 0) and the log file will clear itself.

Optional additions

You can optionally add the:

- ▶ Check BG Link Card Event Handler script
- ▶ Build Log HTML script

This script is meant to be run by Nagios. The sysadmin alerting and notification script is taken care of via Nagios. This script only checks the database and reports its findings.

There are several variables at the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

nagios_ip	The IP address of the Nagios master server
master_log_loc	The location of the directory on the master host where the log file will be stored (in order to view it using the Nagios Web interface)
local_log_loc	The location of the directory on the remote host where the log file is stored
logfile	The name of the log file
loglink	The HTML link (and text) to the log file that will be viewed using the Nagios Web interface. This will be added to the end of all error-state output.

Other things to check

Also, check to:

- ▶ Make sure the DB2 Connection string (found on the "my \$dbh = DBI->connect..." line) has the correct database name, database user, and database password.
- ▶ Make sure the SQL query (found on the "my \$sql = ..." line) has the correct arguments and, most importantly, queries the correct table name.

Usage

To check:

```
./check_bgp_linkcard.pl
```

Return value

The return value is either:

- ▶ Return 0 (OK State in Nagios) if all link cards have a status of "A"
- ▶ Return 2 (CRITICAL State in Nagios) if any link cards have a status that is not "A"

Code

Example B-2 on page 70 is the code to check the BG/P link card.

Example: B-2 Check BG/P link card

```
#!/usr/bin/perl
#####
###
### check_bgp_linkcard.pl
### Created 6/25/2007
###
### Checks the Blue Gene DB2 database. Queries table
### TBGPLINKCARD for any entries with status not
### equal to 'A'. It will return a critical status (exit status
### 2) and give a message of what Link Cards have errored. It
### will also write to a log file, located at
### /var/log/bgmr/nrpe/check_bg_linkcard/linkcard.log.
### If the problem is resolved, and a Link Card's status
### changes back to 'A', then the script will return OK
### status (exit status 0) and the log file will clear
### itself.
###
### This script is meant to be run by Nagios. Sysadmin
### alerting and notification is taken care of via Nagios.
### This script only checks the database and reports its
### findings.
###
#####

#Nagios Master Server
my $nagios_ip = "IP ADDRESS HERE";
#This Host's Linux Hostname
my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#Location of Web-viewable log file on the Nagios Master Server
my $master_log_loc = "WEB DIRECTORY HERE";
#Location of local log file on remote host
my $local_log_loc = "LOG DIRECTORY HERE";
#Name of the Log File
my $logfile = "linkcard.log";
#HTML Link to add to the end of all errored output
my $loglink = "<br><a href='WEB URL HERE/linkcard.html'>Click Here</a> to
view the log file.";

use DBI;
use DBD::DB2::Constants;
use DBD::DB2;
use strict;
#For Nagios
my $TIMEOUT = 15;
my
%ERRORS=('OK'=>0,'WARNING'=>1,'CRITICAL'=>2,'UNKNOWN'=>3,'DEPENDENT'=>4);
#
# DB2 Sourcing
#
my $db2result;
#
$db2result = `./usr/local/bin/db.src`;
#DB2 Connection String
my $dbh = DBI->connect("dbi:DB2:DB_NAME", "DB_USER", "DB_PASSWORD");
#If DB2 connection fails, print error and send CRITICAL ERROR
if(!(defined $dbh))
```



```

{
    print "DB2 Error: $DBI::errstr\n";
    exit 2;
}
#SQL Query
my $sql = "select serialnumber, midplanepos, position, status from
tbgplinkcard where status <> 'A' and status <> 'M' order by serialnumber asc";
my $sth = $dbh->prepare($sql);
$sth->execute();
my ($serialnumber, $midplanepos, $position, $status);
my $count = 0;
my ($output, $shortoutput);
my ($serialleft, $serialright);
#Fetch DB2 data row by row
while ( ($serialnumber, $midplanepos, $position, $status) =
$sth->fetchrow() )
{
    $count = $count + 1;
    #Remove Leading & Trailing White Space
    $serialnumber =~ s/^\s+//;
    $serialnumber =~ s/\s+$//;
    $midplanepos =~ s/^\s+//;
    $midplanepos =~ s/\s+$//;
    $position =~ s/^\s+//;
    $position =~ s/\s+$//;
    $status =~ s/^\s+//;
    $status =~ s/\s+$//;
    $output .= "Link Card failed: Pos=$midplanepos-$position
Status=$status <br><br> ";
}
$sth->finish();
$dbh->disconnect();
#If there are errors to print, write them to the log file.
open(MYOUTFILE, ">$local_log_loc/$logfile");
my $fileoutput = $output;
$fileoutput =~ s/<br><br>/\n/g;
print MYOUTFILE $fileoutput;
close(MYOUTFILE);
#SCP the Log File to the correct place on the Nagios Master Server,
#and set the permissions correctly.

#YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is
#to send a copy of the log file to the Nagios master server, put it in a
#folder under the Apache Web Directory, then change the permissions
#and the owner to be the nagios user and nagios group.
system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@${nagios_ip}:~USERID' >
/dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo mkdir -p $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");

```

```

system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc/$logfile \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null");
#If there are errors, print them and send a CRITICAL status.
if ($output ne "")
{
    #If full output is longer than nagios allows,
    #Use the short output.
    if( length($output) + length($loglink) > 320 )
    {
        $output = "$count Link Cards Failed $loglink";
    }
    else
    {
        $output = substr($output, 0, - 9);
        $output .= $loglink;
    }
    print( $output, "\n" );
    exit $ERRORS{"CRITICAL"};
}
else
{
    print "OK : No Link Cards Failed\n";
    exit $ERRORS{"OK"};
}

```

B.3 Check BG/P midplane

This script checks the Blue Gene DB2 database. It queries table TBGPMIDPLANE for any entries with status not equal to “A”. It will return a critical status (exit status 2) and give a message indicating which midplanes experienced errors. It will also write to a log file, which is located at `/var/log/bgmr/nrpe/check_bg_linkcard/midplane.log`. If the problem is resolved and a midplane’s status changes back to “A”, the script will return OK status (exit status 0) and the log file will clear itself.

Optional additions

You can optionally run the:

- ▶ Check BG Midplane Event Handler script
- ▶ Build Log HTML script

This script is meant to be run by Nagios. The sysadmin alerting and notification script is taken care of via Nagios. This script only checks the database and reports its findings.

There are several variables at the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

nagios_ip	The IP address of the Nagios master server
master_log_loc	The location of the directory on the master host where the log file will be stored (in order to view it using the Nagios Web interface)

local_log_loc	The location of the directory on the remote host where the log file is stored
logfile	The name of the log file
loglink	The HTML link (and text) to the log file that will be viewed using the Nagios Web interface. This will be added to the end of all error-state output.

Other things to check

Check to:

- ▶ Make sure the DB2 Connection string (found on the “my \$dbh = DBI->connect...” line) has the correct database name, database user, and database password.
- ▶ Make sure the SQL query (found on the “my \$sql = ...” line) has the correct arguments and most importantly queries the correct table name.

Usage

To check the usage:

```
./check_bgp_midplane.pl
```

Return value

The return value can be either:

- ▶ Return 0 (OK State in Nagios) if all midplanes have a status of “A”
- ▶ Return 2 (CRITICAL State in Nagios) if any midplanes have a status that is not “A”

Code

Example B-3 is the code to check the BG/P midplane.

Example: B-3 Check BG/P midplane

```
#!/usr/bin/perl
#####
###
### check_bgp_midplane.pl
### Created 6/25/2007
###
### Checks the Blue Gene DB2 database. Queries table
### TBGPMIDPLANE for any entries with status not
### equal to 'A'. It will return a critical status (exit status
### 2) and give a message of what Midplanes have errored. It
### will also write to a log file, located at
### /var/log/bgmr/nrpe/check_bg_midplane/midplane.log.
### If the problem is resolved, and a Midplane's status
### changes back to 'A', then the script will return OK
### status (exit status 0) and the log file will clear
### itself.
###
### This script is meant to be run by Nagios. Sysadmin
### alerting and notification is taken care of via Nagios.
### This script only checks the database and reports its
### findings.
###
#####
#Nagios Master Server
```

```

my $nagios_ip = "IP ADDRESS HERE";
#This Host's Linux Hostname
my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#Location of Web-viewable log file on the Nagios Master Server
my $master_log_loc = "WEB DIRECTORY HERE";
#Location of local log file on remote host
my $local_log_loc = "LOG DIRECTORY HERE";
#Name of the Log File
my $logfile = "midplane.log";
#HTML Link to add to the end of all errored output
my $loglink = "<br><a href='WEB URL HERE/midplane.html'>Click Here</a> to
view the log file.";
use DBI;
use DBD::DB2::Constants;
use DBD::DB2;
use strict;
#For Nagios
my $TIMEOUT = 15;
my
%ERRORS=('OK'=>0,'WARNING'=>1,'CRITICAL'=>2,'UNKNOWN'=>3,'DEPENDENT'=>4);
#      #DB2 Sourcing
#      my $db2result;
#      $db2result = `./usr/local/bin/db.src`;

#DB2 Connection String
my $dbh = DBI->connect("dbi:DB2:DB_NAME", "DB_USER", "DB_PASSWORD");
#If DB2 connection fails, print error and send CRITICAL ERROR
if(!(defined $dbh))
{
    print "DB2 Error: $DBI::errstr\n";
    exit 2;
}
#SQL Query
my $sql = "select serialnumber, posinmachine, status, ismaster from
tbpgmidplane where status <> 'A' and status <> 'M' order by serialnumber asc";
my $sth = $dbh->prepare($sql);
$sth->execute();
my ($serialnumber, $posinmachine, $status, $ismaster);
my $count = 0;
my ($output, $shortoutput);
my ($serialleft, $serialright);
#Fetch DB2 data row by row
while ( ($serialnumber, $posinmachine, $status, $ismaster) =
$sth->fetchrow() )
{
    $count = $count + 1;
    #Remove Leading & Trailing White Space
    $serialnumber =~ s/^\s+//;
    $serialnumber =~ s/\s+$//;
    $posinmachine =~ s/^\s+//;
    $posinmachine =~ s/\s+$//;
    $status =~ s/^\s+//;
    $status =~ s/\s+$//;
    $ismaster =~ s/^\s+//;

```

```

        $ismaster =~ s/\s+$/;
        $output .= "Midplane failed: Pos=$posinmachine Status=$status
IsMaster=$ismaster <br><br> ";
    }
    $sth->finish();
    $dbh->disconnect();
    #If there are errors to print, write them to the log file.
    open(MYOUTFILE, ">$local_log_loc/$logfile");
    my $fileoutput = $output;
    $fileoutput =~ s/<br><br>/\n/g;
    print MYOUTFILE $fileoutput;
    close(MYOUTFILE);
    #SCP the Log File to the correct place on the Nagios Master Server,
    #and set the permissions correctly.

    #YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is
    #to send a copy of the log file to the Nagios master server, put it in a
    #folder under the Apache Web Directory, then change the permissions
    #and the owner to be the nagios user and nagios group.
    system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@${nagios_ip}:~USERID' >
/dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo mkdir -p $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc/$logfile \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null");
    #If there are errors, print them and send a CRITICAL status.
    if ($output ne "")
    {
        #If full output is longer than Nagios allows,
        #Use the short output.
        if( length($output) + length($loglink) > 320 )
        {
            $output = "$count Midplanes Failed $loglink";
        }
        else
        {
            $output = substr($output, 0, - 9);
            $output .= $loglink;
        }
        print( $output, "\n" );
        exit $ERRORS{"CRITICAL"};
    }
    else
    {
        print "OK : No Midplanes Failed\n";
        exit $ERRORS{"OK"};
    }

```

}

B.4 Check BG/P node

This script checks the Blue Gene DB2 database. It queries table TBGPNODE for any entries with status not equal to “A”. It will return a critical status (exit status 2) and give a message indicating which nodes experienced errors. It will also write to a log file, which is located at `/var/log/bgmr/nrpe/check_bg_node/node.log`. If the problem is resolved and a node’s status changes back to “A”, the script will return OK status (exit status 0) and the log file will clear itself.

Optional additions

You can also run the:

- ▶ Check BG Node Event Handler script
- ▶ Build Log HTML script

This script is meant to be run by Nagios. The sysadmin alerting and notification script is taken care of via Nagios. This script only checks the database and reports its findings.

There are several variables at the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

nagios_ip	The IP address of the Nagios master server
master_log_loc	The location of the directory on the master host where the log file will be stored (in order to view it using the Nagios Web interface)
local_log_loc	The location of the directory on the remote host where the log file is stored
logfile	The name of the log file
loglink	The HTML link (and text) to the log file that will be viewed using the Nagios Web interface. This will be added to the end of all error-state output.

Other things to check

Be sure that you:

- ▶ Make sure the DB2 Connection string (found on the “my \$dbh = DBI->connect...” line) has the correct database name, database user, and database password.
- ▶ Make sure the SQL query (found on the “my \$sql = ...” line) has the correct arguments and, most importantly, queries the correct table name.

Usage

To check usage:

```
./check_bgp_node.pl
```

Return value

The return value is either:

- ▶ Return 0 (OK State in Nagios) if all nodes have a status of “A”
- ▶ Return 2 (CRITICAL State in Nagios) if any nodes have a status that is not “A”

Code

Example B-4 is the code to check the BG/P node.

Example: B-4 Check BG/P node

```
#!/usr/bin/perl
#####
###
### check_bgp_node.pl
### Created 6/25/2007
###
### Checks the Blue Gene DB2 database. Queries table
### TBGPNODE for any entries with status not
### equal to 'A'. It will return a critical status (exit status
### 2) and give a message of what Nodes have errored. It
### will also write to a log file, located at
### /var/log/bgmr/nrpe/check_bg_node/node.log.
### If the problem is resolved, and a Node's status
### changes back to 'A', then the script will return OK
### status (exit status 0) and the log file will clear
### itself.
###
### This script is meant to be run by Nagios. Sysadmin
### alerting and notification is taken care of via Nagios.
### This script only checks the database and reports its
### findings.
###
#####
#Nagios Master Server
my $nagios_ip = "IP ADDRESS HERE";
#This Host's Linux Hostname
my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#Location of Web-viewable log file on the Nagios Master Server
my $master_log_loc = "WEB DIRECTORY HERE";
#Location of local log file on remote host
my $local_log_loc = "LOG DIRECTORY HERE";
#Name of the Log File
my $logfile = "node.log";
#HTML Link to add to the end of all errored output
my $loglink = "<br><a href='WEB URL HERE/node.html'>Click Here</a> to view
the log file.";
use DBI;
use DBD::DB2::Constants;
use DBD::DB2;
use strict;
#For Nagios
my $TIMEOUT = 15;
my
%ERRORS=('OK'=>0,'WARNING'=>1,'CRITICAL'=>2,'UNKNOWN'=>3,'DEPENDENT'=>4);
#
# DB2 Sourcing
# my $db2result;
# $db2result = `./usr/local/bin/db.src`;
#DB2 Connection String
my $dbh = DBI->connect("dbi:DB2:DB_NAME", "DB_USER", "DB_PASSWORD");
#If DB2 connection fails, print error and send CRITICAL ERROR
```

```

if(!(defined $dbh))
{
    print "DB2 Error: $DBI::errstr\n";
    exit 2;
}
#SQL Query
my $sql = "select serialnumber, midplanepos, nodecardpos, position, status
from tbgnode where status <> 'A' and status <> 'M'";
my $sth = $dbh->prepare($sql);
$sth->execute();
my ($serialnumber, $midplanepos, $nodecardpos, $position, $status);
my $count = 0;
my ($output, $shortoutput);
my ($serialleft, $serialright);
#Fetch DB2 data row by row
while ( ($serialnumber, $midplanepos, $nodecardpos, $position, $status) =
$sth->fetchrow() )
{
    $count = $count + 1;
    #Remove Leading & Trailing White Space
    $serialnumber =~ s/^\s+//;
    $serialnumber =~ s/\s+$//;
    $midplanepos =~ s/^\s+//;
    $midplanepos =~ s/\s+$//;
    $nodecardpos =~ s/^\s+//;
    $nodecardpos =~ s/\s+$//;
    $position =~ s/^\s+//;
    $position =~ s/\s+$//;
    $status =~ s/^\s+//;
    $status =~ s/\s+$//;
    $output .= "Node failed: Status=$status
Pos=$midplanepos-$nodecardpos-$position <br><br> ";
}
$sth->finish();
$dbh->disconnect();
#If there are errors to print, write them to the log file.
open(MYOUTFILE, ">$local_log_loc/$logfile");
my $fileoutput = $output;
$fileoutput =~ s/<br><br>/\n/g;
print MYOUTFILE $fileoutput;
close(MYOUTFILE);
#SCP the Log File to the correct place on the Nagios Master Server,
#and set the permissions correctly.

#YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is
#to send a copy of the log file to the Nagios master server, put it in a
#folder under the Apache Web Directory, then change the permissions
#and the owner to be the nagios user and nagios group.
system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@${nagios_ip}:~USERID' >
/dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo mkdir -p $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc \"' > /dev/null");

```



```

        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc/$logfile \"' > /dev/null");
        system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null");
        #If there are errors, print them and send a CRITICAL status.
        if ($output ne "")
        {
            #If full output is longer than Nagios allows,
            #Use the short output.
            if( length($output) + length($loglink) > 320 )
            {
                $output = "$count Nodes Failed $loglink";
            }
            else
            {
                $output = substr($output, 0, - 9);
                $output .= $loglink;
            }
            print( $output, "\n" );
            exit $ERRORS{"CRITICAL"};
        }
        else
        {
            print "OK : No Nodes Failed\n";
            exit $ERRORS{"OK"};
        }
}

```

B.5 Check BG/P node card

This script checks the Blue Gene DB2 database. It queries table TBGPNODECARD for any entries with status not equal to "A". It will return a critical status (exit status 2) and give a message indicating which node cards experienced errors. It will also write to a log file, which is located at `/var/log/bgmr/nrpe/check_bg_nodecard/nodecard.log`. If the problem is resolved, and a node card's status changes back to "A", the script will return OK status (exit status 0) and the log file will clear itself.

Optional additions

You can optionally add the:

- ▶ Check BG Node Card Event Handler script
- ▶ Build Log HTML script

This script is meant to be run by Nagios. The sysadmin alerting and notification script is taken care of via Nagios. This script only checks the database and reports its findings.

There are several variables at the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

nagios_ip	The IP address of the Nagios master server
master_log_loc	The location of the directory on the master host where the log file will be stored (in order to view it using the Nagios Web interface)
local_log_loc	The location of the directory on the remote host where the log file is stored
logfile	The name of the log file
loglink	The HTML link (and text) to the log file that will be viewed using the Nagios Web interface. This will be added to the end of all error-state output

Other things to check

Be sure you:

- ▶ Make sure the DB2 Connection string (found on the “my \$dbh = DBI->connect...” line) has the correct database name, database user, and database password.
- ▶ Make sure the SQL query (found on the “my \$sql = ...” line) has the correct arguments and, most importantly, queries the correct table name.

Usage

The command to check:

`./check_bgp_nodecard.pl`

Return value

The return value is either:

- ▶ Return 0 (OK State in Nagios) if all node cards have a status of “A”
- ▶ Return 2 (CRITICAL State in Nagios) if any node cards have a status that is not “A”

Code

Example B-5 is the code to check the BG/P node card.

Example: B-5 Check BG/P node card

```
#!/usr/bin/perl
#####
###
### check_bgp_nodecard.pl
### Created 6/25/2007
###
### Checks the Blue Gene DB2 database. Queries table TBGPNODECARD for any entries
### with status not equal to 'A'. It will return a critical status (exit status 2)
### and give a message of what Node Cards have errored. It will also write to a
### log file, located at /var/log/bgmr/nrpe/check_bg_nodecard/nodecard.log.
### If the problem is resolved, and a Node Card's status changes back to 'A', then
### the script will return OK status (exit status 0) and the log file will clear
### itself.
###
### This script is meant to be run by Nagios. Sysadmin alerting and notification
### is taken care of via Nagios. This script only checks the database and reports
### its findings.
```

```
#####
#Nagios Master Server
my $nagios_ip = "IP ADDRESS HERE";
#This Host's Linux Hostname
my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#Location of Web-viewable log file on the Nagios Master Server
my $master_log_loc = "WEB DIRECTORY HERE";
#Location of local log file on remote host
my $local_log_loc = "LOG DIRECTORY HERE";
#Name of the Log File
my $logfile = "nodecard.log";
#HTML Link to add to the end of all errored output
my $loglink = "<br><a href='WEB URL HERE/nodecard.html'>Click Here</a> to
view the log file.";
use DBI;
use DBD::DB2::Constants;
use DBD::DB2;
use strict;
#For Nagios
my $TIMEOUT = 15;
my
%ERRORS=('OK'=>0,'WARNING'=>1,'CRITICAL'=>2,'UNKNOWN'=>3,'DEPENDENT'=>4);
#
#DB2 Sourcing
#
my $db2result;
#
$db2result = `./usr/local/bin/db.src`;
#DB2 Connection String
my $dbh = DBI->connect("dbi:DB2:DB_NAME", "DB_USER", "DB_PASSWORD");
#If DB2 connection fails, print error and send CRITICAL ERROR
if(!(defined $dbh))
{
    print "DB2 Error: $DBI::errstr\n";
    exit 2;
}
#SQL Query
my $sql = "select serialnumber, midplanepos, position, status from
tbgnodecard where status <> 'A' and status <> 'M'";
my $sth = $dbh->prepare($sql);
$sth->execute();
my ($serialnumber, $midplanepos, $position, $status);
my $count = 0;
my ($output, $shortoutput);
my ($serialleft, $serialright);
#Fetch DB2 data row by row
while ( ($serialnumber, $midplanepos, $position, $status) =
$sth->fetchrow() )
{
    $count = $count + 1;
    #Remove Leading & Trailing White Space
    $serialnumber =~ s/^\s+//;
    $serialnumber =~ s/\s+$//;
    $midplanepos =~ s/^\s+//;
    $midplanepos =~ s/\s+$//;
    $position =~ s/^\s+//;
    $position =~ s/\s+$//;
}
```

```

        $status =~ s/^\s+//;
        $status =~ s/\s+$//;
        $output .= "Node Card failed: Status=$status
Pos=$midplanepos-$position <br><br> ";
    }
    $sth->finish();
    $dbh->disconnect();
    #If there are errors to print, write them to the log file.
    open(MYOUTFILE, ">/var/log/bgmr/nrpe/check_bg_nodocard/nodocard.log");
    my $fileoutput = $output;
    $fileoutput =~ s/<br><br>/\n/g;
    print MYOUTFILE $fileoutput;
    close(MYOUTFILE);
    #SCP the Log File to the correct place on the Nagios Master Server,
    #and set the permissions correctly.

    #YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is
    #to send a copy of the log file to the Nagios master server, put it in a
    #folder under the Apache Web Directory, then change the permissions
    #and the owner to be the nagios user and nagios group.
    system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@${nagios_ip}:~USERID' >
/dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo mkdir -p $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc/$logfile \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null");
    #If there are errors, print them and send a CRITICAL status.
    if ($output ne "")
    {
        #If full output is longer than Nagios allows,
        #Use the short output.
        if( length($output) + length($loglink) > 320 )
        {
            $output = "$count Node Cards Failed $loglink";
        }
        else
        {
            $output = substr($output, 0, - 9);
            $output .= $loglink;
        }
        print( $output, "\n" );
        exit $ERRORS{"CRITICAL"};
    }
    else
    {
        print "OK : No Node Cards Failed\n";
    }

```

```
        exit $ERRORS{"OK"};
    }
}
```

B.6 Check BG/P service action

This script checks the Blue Gene DB2 database. It queries table TBGPSERVICEACTION for any entries with status not equal to "C". It will return a critical status (exit status 2) and give a message indicating which service actions are unresolved. It will also write to a log file, which is located at `/var/log/bgmr/nrpe/check_bg_serviceaction/serviceaction.log`. If the problem is resolved and a service action's status changes back to "C", the script will return OK status (exit status 0) and the log file will clear itself.

Optional additions

You can optionally add the:

- ▶ Check BG Service Action Event Handler script
- ▶ Build Log HTML script

This script is meant to be run by Nagios. The sysadmin alerting and notification script is taken care of via Nagios. This script only checks the database and reports its findings.

There are several variables at the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

nagios_ip	The IP address of the Nagios master server
master_log_loc	The location of the directory on the master host where the log file will be stored (in order to view it using the Nagios Web interface)
local_log_loc	The location of the directory on the remote host where the log file is stored
logfile	The name of the log file
loglink	The HTML link (and text) to the log file that will be viewed using the Nagios Web interface. This will be added to the end of all error-state output.

Other things to check

Be sure to check:

- ▶ Make sure the DB2 Connection string (found on the "my \$dbh = DBI->connect..." line) has the correct database name, database user, and database password.
- ▶ Make sure the SQL query (found on the "my \$sql = ..." line) has the correct arguments and, most importantly, queries the correct table name.

Usage

To check the usage:

```
./check_bgp_serviceaction.pl
```

Return value

The return value can be either:

- ▶ Return 0 (OK State in Nagios) if all service actions have a status of "A"
- ▶ Return 2 (CRITICAL State in Nagios) if any service actions have a status that is not "A"

Code

Example B-6 is the code to check BG/P service action.

Example: B-6 Check BG/P service action

```
#!/usr/bin/perl
#####
###
### check_bgp_serviceaction.pl
### Created 6/25/2007
###
### Checks the Blue Gene DB2 database. Queries table TBGPSERVICEACTION for any
### entries with status not equal to 'C'. It will return a critical status (exit
### status 2) and give a message of what Service Actions are unresolved. It will
### also write to a log file, located at
### /var/log/bgmr/nrpe/check_bg_serviceaction/serviceaction.log.
### If the problem is resolved, and a Service Action's status changes back to 'C',
### then the script will return OK status (exit status 0) and the log file will
### clear itself.
###
### This script is meant to be run by Nagios. Sysadmin alerting and notification
### is taken care of via Nagios. This script only checks the database and reports
### its findings.
#####
#Nagios Master Server
my $nagios_ip = "IP ADDRESS HERE";
#This Host's Linux Hostname
my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#Location of Web-viewable log file on the Nagios Master Server
my $master_log_loc = "WEB DIRECTORY HERE";
#Location of local log file on remote host
my $local_log_loc = "LOG DIRECTORY HERE";
#Name of the Log File
my $logfile = "serviceaction.log";
#HTML Link to add to the end of all errored output
my $loglink = "<br><a href='WEB URL HERE/serviceaction.html'>Click
Here</a> to view the log file.";
use DBI;
use DBD::DB2::Constants;
use DBD::DB2;
use strict;
#For Nagios
my $TIMEOUT = 15;
my
%ERRORS=('OK'=>0,'WARNING'=>1,'CRITICAL'=>2,'UNKNOWN'=>3,'DEPENDENT'=>4);
#
# DB2 Sourcing
#
my $db2result;
#
$db2result = `./usr/local/bin/db.src`;
#DB2 Connection String
my $dbh = DBI->connect("dbi:DB2:DB_NAME", "DB_USER", "DB_PASSWORD");
#If DB2 connection fails, print error and send CRITICAL ERROR
if(!(defined $dbh))
{
    print "DB2 Error: $DBI::errstr\n";
    exit 2;
}
```

```

    }
    #SQL Query
    my $sql = "select id, location, status, usernameprepareforservice from
    tbgpserviceaction where status <> 'C' order by id asc";
    my $sth = $dbh->prepare($sql);
    $sth->execute();
    my ($id, $location, $status, $usernameprepareforservice);
    my $count = 0;
    my ($output, $shortoutput);
    #Fetch DB2 data row by row
    while ( ($id, $location, $status, $usernameprepareforservice) =
    $sth->fetchrow() )
    {
        $count = $count + 1;
        #Remove Leading & Trailing White Space
        $id =~ s/^\s+//;
        $id =~ s/\s+$//;
        $location =~ s/^\s+//;
        $location =~ s/\s+$//;
        $usernameprepareforservice =~ s/^\s+//;
        $usernameprepareforservice =~ s/\s+$//;
        $status =~ s/^\s+//;
        $status =~ s/\s+$//;
        $output .= "Service Action $id unresolved: Status=$status
    Location=$location Username=$usernameprepareforservice <br><br> ";
    }
    $sth->finish();
    $dbh->disconnect();
    #If there are errors to print, write them to the log file.
    open(MYOUTFILE, ">$local_log_loc/$logfile");
    my $fileoutput = $output;
    $fileoutput =~ s/<br><br>/\n/g;
    print MYOUTFILE $fileoutput;
    close(MYOUTFILE);
    #SCP the Log File to the correct place on the Nagios Master Server,
    #and set the permissions correctly.

    #YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is
    #to send a copy of the log file to the Nagios master server, put it in a
    #folder under the Apache Web Directory, then change the permissions
    #and the owner to be the nagios user and nagios group.
    system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
    ~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@${nagios_ip}:~USERID' >
    /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
    \"sudo mkdir -p $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
    \"sudo chmod 775 $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
    \"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
    \"sudo cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");
    system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
    \"sudo chmod 775 $master_log_loc/$logfile \"' > /dev/null");

```

```

system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null\"");
#If there are errors, print them and send a CRITICAL status.
if ($output ne "")
{
    #If full output is longer than Nagios allows,
    #Use the short output.
    if( length($output) + length($loglink) > 320 )
    {
        $output = "$count Service Actions Unresolved $loglink";
    }
    else
    {
        $output = substr($output, 0, - 9);
        $output .= $loglink;
    }
    print( $output, "\n" );
    exit $ERRORS{"CRITICAL"};
}
else
{
    print "OK : No Service Actions Unresolved\n";
    exit $ERRORS{"OK"};
}

```

B.7 Check BG/P service card

This script checks the Blue Gene DB2 database. It queries table TBGPSEVICECARD for any entries with status not equal to “A”. It will return a critical status (exit status 2) and give a message indicating which service cards experienced errors. It will also write to a log file, which is located at `/var/log/bgmr/nrpe/check_bg_servicecard/servicecard.log`. If the problem is resolved and a service card’s status changes back to “A”, the script will return OK status (exit status 0) and the log file will clear itself.

Optional additions

You can optionally add the:

- ▶ Check BG Service Card Event Handler script
- ▶ Build Log HTML script

This script is meant to be run by Nagios. The sysadmin alerting and notification script is taken care of via Nagios. This script only checks the database and reports its findings.

There are several variables at the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

nagios_ip	The IP address of the Nagios master server
master_log_loc	The location of the directory on the master host where the log file will be stored (in order to view it using the Nagios Web interface)
local_log_loc	The location of the directory on the remote host where the log file is stored
logfile	The name of the log file

loglink The HTML link (and text) to the log file that will be viewed using the Nagios Web interface. This will be added to the end of all error-state output.

Other things to check

Also, check:

- ▶ Make sure the DB2 Connection string (found on the “my \$dbh = DBI->connect...” line) has the correct database name, database user, and database password.
- ▶ Make sure the SQL query (found on the “my \$sql = ...” line) has the correct arguments and, most importantly, queries the correct table name.

Usage

To check usage:

```
./check_bgp_servicecard.pl
```

Return value

The return value is either:

- ▶ Return 0 (OK State in Nagios) if all service cards have a status of “A”
- ▶ Return 2 (CRITICAL State in Nagios) if any service cards have a status that is not “A”

Code

Example B-7 is the code to check the BG/P service card.

Example: B-7 Check BG/P service card

```
#!/usr/bin/perl

#####
###
### check_bg1_servicecard.pl
### Created 6/25/2007
### ###
### Checks the Blue Gene DB2 database. Queries table
### TBGLSERVICECARD for any entries with status not
### equal to 'A'. It will return a critical status (exit status
### 2) and give a message of what Service Cards have errored. It
### will also write to a log file, located at
### /var/log/bgmr/nrpe/check_bg_servicecard/servicecard.log.
### If the problem is resolved, and a Service Card's status
### changes back to 'A', then the script will reutrn OK
### status (exit status 0) and the log file will clear
### itself.
###
### This script is meant to be run by Nagios. Sysadmin
### alerting and notification is taken care of via Nagios.
### This script only checks the database and reports its
### findings.
###
#####

#Nagios Master Server
my $nagios_ip = "IP ADDRESS HERE";
#This Host's Linux Hostname
```

```

my $hostname = `hostname -i`;
$hostname =~ s/\s//g;
#Location of web-viewable log file on the Nagios Master Server
my $master_log_loc = "WEB DIRECTORY HERE";
#Location of local log file on remote host
my $local_log_loc = "LOG DIRECTORY HERE";
#Name of the Log File
my $logfile = "serviceaction.log";
#HTML Link to add to the end of all errored output
my $loglink = "<br><a href='WEB URL HERE/serviceaction.html'>Click
Here</a> to view the log file.";

use DBI;
use DBD::DB2::Constants;
use DBD::DB2;
use strict;

#For Nagios
my $TIMEOUT = 15;
my
%ERRORS=('OK'=>0,'WARNING'=>1,'CRITICAL'=>2,'UNKNOWN'=>3,'DEPENDENT'=>4);

#      #DB2 Sourcing
#      my $db2result;
#      $db2result = `./discovery/db.src`;

#DB2 Connection String
my $dbh = DBI->connect("dbi:DB2:DB_NAME", "DB_USER", "DB_PASSWORD");

#If DB2 connection fails, print error and send CRITICAL ERROR
if(!(defined $dbh))
{
    print "CRITICAL : Error Connecting to DB2 Database:
$DBI::errstr\n";
    exit $ERRORS{'CRITICAL'};
}

#SQL Query
my $sql = "select serialnumber, midplaneserialnumber, posinmidplane,
status, location from tbglservicecard where status <> 'A' order by serialnumber
asc";

my $sth = $dbh->prepare($sql);
$sth->execute();

my ($serialnumber, $midplaneserialnumber, $posinmidplane, $status,
$location);

my $count = 0;
my ($output, $shortoutput);
my ($serialleft, $serialright);

#Fetch DB2 data row by row
while ( ($serialnumber, $midplaneserialnumber, $posinmidplane, $status,
$location) = $sth->fetchrow() )

```

```

{
    $count = $count + 1;

    #Remove Leading & Trailing White Space
    $serialnumber =~ s/^\s+//;
    $serialnumber =~ s/\s+$//;
    $midplaneserialnumber =~ s/^\s+//;
    $midplaneserialnumber =~ s/\s+$//;
    $posinmidplane =~ s/^\s+//;
    $posinmidplane =~ s/\s+$//;
    $status =~ s/^\s+//;
    $status =~ s/\s+$//;
    $location =~ s/^\s+//;
    $location =~ s/\s+$//;

    $output .= "Service Card failed: Pos=$posinmidplane Status=$status
Location=$location <br><br> ";
}

$sth->finish();
$dbh->disconnect();

#If there are errors to print, write them to the log file.
open(MYOUTFILE, ">$local_log_loc/$logfile");

my $fileoutput = $output;
$fileoutput =~ s/<br><br>/\n/g;
print MYOUTFILE $fileoutput;
close(MYOUTFILE);

#SCP the Log File to the correct place on the Nagios Master Server,
#and set the permissions correctly.

#YOU WILL HAVE TO CHANGE THIS TO GET IT WORKING CORRECTLY. The idea is
#to send a copy of the log file to the Nagios master server, put it in a
#folder under the Apache Web Directory, then change the permissions
#and the owner to be the nagios user and nagios group.
system( "sudo su - USERID -c 'sudo scp -o StrictHostKeyChecking=no -i
~USERID/.ssh/id_rsa $local_log_loc/$logfile USERID@${nagios_ip}:~USERID' >
/dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo mkdir -p $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo cp -v ~USERID/$logfile $master_log_loc \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chmod 775 $master_log_loc/$logfile \"' > /dev/null");
system( "sudo su - USERID -c 'ssh ${nagios_ip} -o StrictHostKeyChecking=no
\"sudo chown NAGIOS_USER.NAGIOS_GROUP $master_log_loc/$logfile \"' > /dev/null");

#If there are errors, print them and send a CRITICAL status.

```

```

if ($output ne "")
{
    #If full output is longer than nagios allows,
    #Use the short output.
    if( length($output) + length($loglink) > 320 )
    {
        $output = "$count Service Cards Failed. $loglink";
    }
    else
    {
        $output = substr($output, 0, - 9);
        $output .= $loglink;
    }
    print( $output, "\n" );
    exit $ERRORS{"CRITICAL"};
}
else
{
    print "OK : No Service Cards Failed\n";
    exit $ERRORS{"OK"};
}

```

DB2 plug-ins

This appendix contains the scripts for the DB2 plug-ins.

C.1 DB2 health check

This plug-in checks the health of the DB2 database on a service node. The way this works is by calling the utility `db2diag -l [Warning|Severe,Critical]`, reporting the status of the plug-in, and copying the actual error log file to the management server.

Options are passed on to the script via the command-line that are defined within the NRPE `command.cfg` file (`/usr/bgspeedteam/bgm/dist/nrpe/etc/command.cfg`). If any of these options are incorrect, the script will fail to operate correctly. Verify in the `command.cfg` file (near the end) that the following parameters are correct for the commands:

check_db2_health_warning_BGL, **check_db2_health_warning_BGP**,
check_db2_health_critical_BGL, and **check_db2_health_critical_BGP**:

```
--db2diaguser=[User who can execute db2diag correctly]
--db2diaglog=[Location of correct db2diag.log file]
```

There are several variables towards the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

managementIP	The IP address of the Nagios master server
masterLogDir	The path to the Web-viewable log directory on the Nagios master server (should be located in a folder viewable by Apache)
currentWarningStateFile	The local path to the warning state file used by <code>check_db2_health.pl</code>
currentCriticalStateFile	The local path to the critical state file used by <code>check_db2_health.pl</code>
warningTimestampFile	The local path to the warning timestamp file used by <code>check_db2_health.pl</code>
criticalTimestampFile	The local path to the critical timestamp file used by <code>check_db2_health.pl</code>
warningLog	The local path to the warning log file used by <code>check_db2_health.pl</code>
criticalLog	The local path to the critical log file used by <code>check_db2_health.pl</code>
warningLevel	A warning error level used by the <code>db2diag</code> utility
criticalLevel	A critical error level used by the <code>db2diag</code> utility

In order to use this script, you must create four files related to the plug-in's state. The files are: `lastWarning.timestamp`, `lastCritical.timestamp`, `warning.STATE`, and `critical.STATE`.

In order to do so, issue these commands. You must change the paths specified to something that fits your environment and then change their permission to the nagios user/group:

```
echo "OK" > /path/to/warning.STATE
echo "OK" > /path/to/critical.STATE
echo "FIRSTRUN" > /path/to/lastWarning.timestamp
echo "FIRSTRUN" > /path/to/lastCritical.timestamp
```

Also, prior to running the script, you must specify the variables defined at the top of the script. This includes variables that reference the location of the files that you just created.

Usage

The commands are:

```
./check_db2_health.pl --db2diaguser=[bglsysdb|bgpsysdb|etc] --db2diaglog=[Path to db2diag.log] --alertlevel=[WARNING|CRITICAL]
```

--db2diaguser=[bglsysdb|bgpsysdb|etc] - The user that has the ability to run the “db2diag” utility.

--db2diaglog=[Path to db2diag.log] - The path to the log file that db2diag reads.

--alertlevel=[WARNING|CRITICAL] - the level of alerts for which to check.

Return value

The return values are:

- ▶ OK - If no errors were processed.
- ▶ [*alert level specified*] - If the errors appropriate for the alert level you specified were found. Also gives the URL of the log file.

Code

Example C-1 shows the DB2 health check.

Example: C-1 DB2 health check

```
#!/usr/bin/perl
#####
# File Name: check_db2_health.pl
# Author: Blue Gene Speed Team
# Date: Summer, 2007
# Description: This script calls the db2diag plug-in in order to log all new
# warning and critical errors. It then logs them to a file, copies them over to
# the master host, and outputs error information that is formatted for Nagios.
#
# Usage: ./check_db2_health.pl --alertlevel=[CRITICAL|WARNING]
# Return Value:
#     OK
#     [WARNING|CRITICAL] - Check the log file for more errors by <a
# href="address">clicking here</a>.
#####
# Other:
#####
use Getopt::Long;
# Declare exit code enum
my %EXIT_CODES = (
    'UNKNOWN'    => -1,
    'OK'         => 0,
    'WARNING'    => 1,
    'CRITICAL'   => 2
);

# Variables that need to be modified to fit the current environment
my $managementIP = ""; # IP of management server
my $masterLogDir = ""; # Path to html log directory on management server
my $currentWarningStateFile = ""; # warning state file on this host
my $currentCriticalStateFile = ""; # critical state file on this host
```

```

my $warningTimestampFile = ""; # warning timestamp file on this host
my $criticalTimestampFile = ""; # critical timestamp file on this host
my $warningLog = ""; # warning log file on this host
my $criticalLog = ""; # critical log file on this host
my $warningLevel = "Warning"; # db2diag
level to retrieve for Warnings
my $criticalLevel = "Error,Severe"; # db2diag
level to retrieve for Criticals
# Script variables
my $hostip = `hostname -i`;
my $alertlevel;
my $db2diaguser = "";
my $db2diaglog = "";
my $currentStatusState;
my $numNewErrors;
my $logFile;
my $timestampLine;
# Store command line alertlevel input to var
$result = GetOptions ("alertlevel=s" => \$alertlevel, "db2diaguser=s" =>
\$db2diaguser, "db2diaglog=s" => \$db2diaglog);
# Convert the alertlevel argument to all uppercase.
$alertlevel = uc($alertlevel);
# Check to make sure correct arguments were specified
if (($alertlevel ne "WARNING") && ($alertlevel ne "CRITICAL") || ($db2diaguser eq
"") || ($db2diaglog eq ""))
{
    # Invalid arguments specified, exit.
    print "Invalid check_db2_health parameters.\n";
    print "Correct Usage: ./check_db2_health.pl
--db2diaguser=[bglsysdb|bgpsysdb|etc] --db2diaglog=[Path to db2diag.log]
--alertlevel=[WARNING|CRITICAL]\n";
    exit $EXIT_CODES{"UNKNOWN"};
}
if ( (!-e $db2diaglog) || (!-e $warningTimestampFile) || (!-e
$criticalTimestampFile) || (!-e $currentWarningStateFile) || (!-e
$currentCriticalStateFile)) {
    print "A core file does not exist. Please check the 'check_db2_plugin' on
${hostip}. Unable to function.";
    exit -1;
}
# removes whitespace
$hostip =~ s/\s//g;
# Determine which files/parameters to use for the rest of the program (based on
alertlevel)
if ($alertlevel eq "WARNING")
{
    $timestampFile = $warningTimestampFile;
    $logFile = $warningLog;
    $level = $warningLevel;
    $logFN = "warning";
    $currentStateFile = $currentWarningStateFile;
}
else
{
    $timestampFile = $criticalTimestampFile;

```



```

    $logFile = $criticalLog;
    $level = $criticalLevel;
    $logFN = "critical";
    $currentStateFile = $currentCriticalStateFile;
}
# Extract current status state from state file
open(CURRENTSTATE, $currentStateFile);
$currentStatusState = <CURRENTSTATE>;
chomp($currentStatusState);
close(CURRENTSTATE);
# Read in timestamp
open(TIMESTAMP, $timestampFile);          # Open the file
$timestampLine = <TIMESTAMP>;             # Read it into an array
chomp($timestampLine);                    # chomp it
close(TIMESTAMP);                         # Close the file
if ($timestampLine eq "FIRSTRUN") {
    # If this is the first run of the script, set the timestamp to a week
    since today
    $tempTimestamp=`date +%Y-%m-%d-%k.%M.%S.%N`;
    chomp($tempTimestamp);
    $tempTimestamp = substr($tempTimestamp, 0, -5);
    open bCURRENTSTATE, '>', ${timestampFile} or die "Can't open ${logFile}
$!";
    print bCURRENTSTATE $tempTimestamp;
    close(bCURRENTSTATE);
    $timestampLine = $tempTimestamp;
}
# Set permissions correctly on log file
system("sudo chown ${db2diaguser}.nagcmd $logFile | sudo chmod 775 $logFile &>
/dev/null");
# get all errors since last timestamp
system("sudo su - ${db2diaguser} -c 'db2diag -l $level -time $timestampLine
$db2diaglog >> $logFile'");
$numNewErrors = `sudo su - ${db2diaguser} -c 'db2diag -l $level -time
$timestampLine $db2diaglog | wc -l`";
# Set permissions correctly on log file
system("sudo su - ${db2diaguser} -c 'chown ${db2diaguser}.nagcmd $logFile &>
/dev/null'");
# if in OK state and new errors were processed, change state to $alertstate.
if ($numNewErrors > 0)
{
    # Store timestamp of last entry if there were new entries
    system("sudo su - ${db2diaguser} -c 'db2diag -l $level -fmt %timestamp
-time $timestampLine $db2diaglog | tail --lines=1 > $timestampFile'");
    if ($currentStatusState eq "OK")
    {
        # Change the current status state to the alert state
        $currentStatusState = $alertlevel;
        # Write current status state to the state file
        open CURRENTSTATE, ">$currentStateFile" or die "Can't open
$currentStateFile $!";
        print CURRENTSTATE "$currentStatusState";
        close(CURRENTSTATE);
        # Copy files over to the management node

```

```

        system("sudo su - bgsetup -c 'sudo scp -i ~bgsetup/.ssh/id_rsa
$logFile bgsetup\@${managementIP}:~bgsetup' > /dev/null" );
        system("sudo su - bgsetup -c 'ssh $managementIP \"sudo mkdir -p
${masterLogDir}/${hostip}/check_db2_health\"' &> /dev/null");
        system("sudo su - bgsetup -c 'ssh $managementIP \"sudo cp -v
~bgsetup/${logFN}.log ${masterLogDir}/${hostip}/check_db2_health\"' &>
/dev/null");
        system( "sudo su - bgsetup -c 'ssh $managementIP \"sudo chmod -R
775 ${masterLogDir}/${hostip}/check_db2_health\"' > /dev/null");
        system( "sudo su - bgsetup -c 'ssh $managementIP \"sudo chown -R
nagios.nagcmd ${masterLogDir}/${hostip}/check_db2_health\"' > /dev/null");
    }
}
# exit the program and print the right output for Nagios
if ($currentStatusState eq 'OK') {
    print "OK\n";
    exit $EXIT_CODES{"OK"};
}
else {
    print "$currentStatusState - <a
href=\"../logs/${hostip}/check_db2_health/${logFN}.html\">Click Here</a> to view
the log file.";
    exit $EXIT_CODES{"${currentStatusState}"};
}

```



Other plug-ins

This appendix contains several plug-ins.

D.1 Check daemon

This script checks to see if a daemon is running. Given a process through the command line, the plug-in does a **ps** command and **greps** on the process you specify. If the number of lines returned by the **ps** command is fewer than the number you specify in the command line, the plug-in returns a CRITICAL status. This plug-in is meant to be run by Nagios or remotely by NRPE.

There are several hard-coded considerations when implementing this plug-in. One consideration is to add the full path to the **ps**, **grep**, or **wc** Linux utilities if they are not recognized.

Usage

The usage is:

```
check_daemon.pl -p <PROCESS> -l <# LINES>
```

<PROCESS> The process for which to check. You can use the full path name for accuracy or a single process name.

<# LINES> The number of lines (number of processes returned by **ps**) for which to check. If the command returns fewer than this number, the plug-in will return a CRITICAL status.

Return value

The return value can be:

- ▶ 0 - OK Status - The **ps** command returns equal or more than **<# LINES>** processes that match **<PROCESS>**.
- ▶ 2 - CRITICAL Status - The **ps** command returns less than **<# LINES>** processes that match **<PROCESS>**.

Code

Example D-1 is the check daemon plug-in.

Example: D-1 Check daemon

```
#!/usr/bin/perl
#####
# File Name: check_daemon.pl
# Author: Blue Gene Speed Team
# Date: Summer, 2007
# Description: Checks to see if a daemon is running. Given a process through the
#               command line,
#               the plug-in does a ps command and greps on the process you specify. If the
#               number of lines
#               returned by the ps command is fewer than the number you specify in the
#               command line,
#               the pluin returns a CRITICAL status. This plug-in is meant to be run by
#               Nagios,
#               or remotely by NRPE.
# Usage: check_daemon.pl -p <PROCESS> -l <# LINES>
#       <PROCESS> - The process to check for. Can use full pathname for accuracy
#       or a single
#               process name.
```

```

#      <# LINES> - The number of lines (number of processes returned by ps) to
check for. If
#      the command returns fewer than this amount, the plug-in will
return a CRITICAL status.
# Return Value:
#      0 - OK Status - The ps command returns equal or more than <# LINES>
processes that match <PROCESS>
#      2 - CRITICAL Status - The ps command returns less than <# LINES> processes
that match <PROCESS>
#####
# Other: There is an issue with the Nagios daemon truncating output of the PS
command. A process with a long argument list or pathname may not all appear when
the command is run. If Nagios returns a CRITICAL status from this plug-in that you
know to be incorrect (a process is running when Nagios says it isn't), this may be
the problem. We haven't determined a good solution yet, but uncommenting the
"stty" line below may help.
#####
use Getopt::Long;
my $process;
my $output = "";
my $checklines=0;
my $result = GetOptions("p=s" => \$process, "l=i" => \$checklines);
if($process eq "" || $checklines == 0)
{
    print "Usage: check_daemon -p <PROCESS> -l <# LINES>\n";
    print "check_daemon does a ps command, searching for <PROCESS>.\n";
    print "It counts the number of lines returned by ps. ";
    print "If that is less than <# LINES>, then check_daemon returns
critical.\n";
    exit 0;
}
#Uncomment this line if you are having problems mentioned in the Other section.
#my $result = `stty columns 32`;
#Find how many processes match the given.
my $lines = `ps aux | grep -v grep | grep -v check_daemon.pl | grep $process | wc
-l`;
my $check = `ps aux | grep -v grep | grep -v check_daemon.pl | grep $process`;
while($check ne "")
{
    #65 Characters in is the start of the process name column.
    $check =~ m/{65}(.)\n/;
    #Add the process name to the output.
    $output .= "$1 <br> ";
    #We are finished with that line - remove it.
    $check =~ s/.*\n//;
}
$output =~ s/ <br> $//;
#Parse out the path and just include the filename
if($process =~ m/\./\.(.*)$/)
{
    $process = $1;
}
#If fewer processes are found than specified...
if($lines < $checklines)
{

```

```

        if( $output != m/^$/ )
        {
            print "$process CRITICAL: Daemon not running correctly: <br>
$output";
            exit 2; #Exit with CRITICAL status
        }
        else
        {
            print "$process CRITICAL: Daemon not running correctly: No
Processes Found\n";
            exit 2; #Exit with CRITICAL status
        }
    }
    else
    {
        print "$process OK: Daemon running correctly.";
        exit 0; #Exit with OK Status
    }
}

```

D.2 Who users

This is a Nagios plug-in that outputs the users logged in on a host.

Usage

The usage is:

./who_users.pl

Return value

The return value is:

0 - OK - Always OK State. No other output.

Code

Example D-2 shows the Who users plug-in.

Example: D-2 Who users

```

#!/usr/bin/perl
#####
# File Name: who_users.pl
# Author: Blue Gene Speed Team, Summer 2007
# Date: 7/25/2007
# Description: Nagios plug-in that outputs the users logged in on a host.
# Usage: ./who_users.pl
# Return Value:
#     0 - OK - Always OK State. No other output.
#####
# Other:
#####
my $output;
$output = `who -q | head --lines 1`;
print "Users: ", $output;

```

```
exit 0;
```

Archived

Archived



Plug-in resources

This appendix contains plug-in resources.

E.1 Event handlers

Event handlers are plug-in resources.

E.1.1 Event handler template

This event handler template is a base for all Nagios services that require that errors are stored in a log file and presented to the user in an html file. There are several hard-coded aspects to consider when implementing this script:

- ▶ Variables declared at the top of the file, such as `pluginName`, `logName`, `htmlName`, `relativeLogPath`, and `htmlFile`
- ▶ Path to the `build_html.pl` plug-in
- ▶ Any paths within the OK conditional
- ▶ Any arguments within the OK conditional

In reference to `"/usr/lib/nagios/plugins/check_nrpe -H "$ARGV[3]" -c check_db2_health_critical_reset`;` The `'check_db2_health_critical_reset'` needs to be changed if you modify the NRPE command file and rename the command.

Usage

The usage is:

```
ARGV[0]: Service State ($SERVICESTATE$ within Nagios)
ARGV[1]: Hostname ($HOSTNAME$ within Nagios)
ARGV[2]: Service Name ($SERVICEDESC$ within Nagios)
ARGV[3]: Host Address ($HOSTADDRESS$ within Nagios)
```

Return value

The value returned is 0.

Code

Example E-1 is the event handler template.

Example: E-1 Event handler template

```
#!/usr/bin/perl
#####
#
# File Name: event_handler_template.pl
# Author: Blue Gene Speed Team
# Date: Summer, 2007
#
# Description: This event handler template is a base for all Nagios services that
# require that errors be stored in a log file, and presented to the user in an html
# file.
# Parameters:
# ARGV[0]: Service State ($SERVICESTATE$ within Nagios)
# ARGV[1]: Hostname ($HOSTNAME$ within Nagios)
# ARGV[2]: Service Name ($SERVICEDESC$ within Nagios)
# ARGV[3]: Host Address ($HOSTADDRESS$ within Nagios)
#####
```

```

# Other: This script is triggered when a Nagios service enters a new state.
#Conditions that this script processes by default are OK, WARNING, CRITICAL, and
#UNKNOWN. Upon a warning or critical state, the file build_html.pl will be called
#with the correct arguments and it will create an html file for viewing log files
#and making external commands on a service.
#####
# Edit the below variables as needed for your plug-in.
my $pluginName = "INSERT PLUGIN NAME HERE";           # ie 'check_ping'
my $logName = "INSERT LOG FILE NAME HERE";           # ie criticalerror.log
my $htmlName = "INSERT HTML FILE NAME HERE";         # ie criticalerror.html
my $relativeLogPath = "RELATIVE URL LOG PATH";       # Web path of the
log files
my $htmlFile="PHYSICAL LOG PATH";                    # Log directory within the html
directory on your server
my $build_html_Location = "NAGIOS PLUGINS DIR/build_html.pl"; # Location of
the build_html.pl file for building Plugins
# Construct $relativeLogPath and $htmlFile using the IP address to delineate which
host the files belong too
$relativeLogPath = $relativeLogPath . $ARGV[3] . "/" . $pluginName . "/" .
$logName;
$htmlFile = $htmlFile . $ARGV[3] . "/" . $pluginName . "/" . $htmlName;
# What state is the service in?
if($ARGV[0] eq "OK") #What to do in the OK State...
{
}
elsif($ARGV[0] eq "WARNING") #What to do in the WARNING State...
{
    system("${build_html_Location} \"$ARGV[1]\" \"$ARGV[2]\" $relativeLogPath
$htmlFile");
}
elsif($ARGV[0] eq "UNKNOWN") #What to do in the UNKNOWN State...
{
}
elsif($ARGV[0] eq "CRITICAL") #What to do in the CRITICAL State...
{
    system("${build_html_Location} \"$ARGV[1]\" \"$ARGV[2]\" $relativeLogPath
$htmlFile");
}
exit 0;

```

E.1.2 Check Blue Gene (BG) jobs event handler

This event handler runs on a state change for the BG Job History service. On an OK state, it will call `check_nrpe` and Reset all job errors for that host. On a WARNING or CRITICAL state, it will call `build_html.pl` to build an HTML file that will help in viewing the plug-in's log file using Nagios. There are several hard-coded aspects to consider when implementing this script:

- ▶ Variables declared at the top of the file, such as `pluginName`, `logName`, `htmlName`, `relativeLogPath`, and `htmlFile`
- ▶ Path to the `build_html.pl` plug-in
- ▶ Any paths within the OK conditional
- ▶ Any arguments within the OK conditional

In reference to `"/usr/lib/nagios/plugins/check_nrpe -H "$ARGV[3]" -c check_db2_health_critical_reset";` The `'check_db2_health_critical_reset'` needs to be changed if you modify the NRPE command file and rename the command.

Usage

To use:

ARGV[0]: Service State (\$SERVICESTATE\$ within Nagios)
ARGV[1]: Hostname (\$HOSTNAME\$ within Nagios)
ARGV[2]: Service Name (\$SERVICEDESC\$ within Nagios)
ARGV[3]: Host Address (\$HOSTADDRESS\$ within Nagios)

Return value

The return value is 0.

Code

Example E-2 is the check BG jobs event handler.

Example: E-2 Check BG jobs event handler

```
#!/usr/bin/perl
#####
#
# File Name: check_bg_jobs_event.pl
# Author: Blue Gene Speed Team, Summer 2007
# Date: 7/24/2007
# Description: This event handler runs on a state change for the BG Job History
#service. On an OK state, it will call check_nrpe and Reset all job errors for
#that host. On a WARNING or CRITICAL state, it will call build_html.pl to build an
#HTML file that will help in viewing the plug-in's log file using Nagios.
# Parameters:
# ARGV[0]: Service State ($SERVICESTATE$ within Nagios)
# ARGV[1]: Hostname ($HOSTNAME$ within Nagios) #
# ARGV[2]: Service Name ($SERVICEDESC$ within Nagios) #
# ARGV[3]: Host Address ($HOSTADDRESS$ within Nagios) #
#####
# Other: This script is triggered when BG Job History service enters a new state.
#Conditions that this script process by default are OK, WARNING, CRITICAL, and
#UNKNOWN. Upon a warning or critical state, the file build_html.pl will be called
#with the correct arguments and it will create an html file for viewing log files
#and making external commands on a service. #
#####
# Edit the below variables as needed for your plug-in.
my $pluginName = "check_bg_jobs"; # ie 'check_ping'
my $logName = "joberrors.log"; # ie criticalerror.log
my $htmlName = "joberrors.html"; # ie criticalerror.html
my $relativeLogPath = "RELATIVE URL LOG PATH"; # Web path of the
log files
my $htmlFile="PHYSICAL LOG PATH"; # Log directory within the html
directory on your server
# Construct $relativeLogPath and $htmlFile using the IP address to delineate which
host the files belong too
$relativeLogPath = $relativeLogPath . $ARGV[3] . "/" . $pluginName . "/" .
$logName;
$htmlFile = $htmlFile . $ARGV[3] . "/" . $pluginName . "/" . $htmlName;
```

```
# What state is the service in?
if($ARGV[0] eq "OK") #What to do in the OK State...
{
    `NAGIOS PLUGINS DIRECTORY/check_nrpe -H $ARGV[3] -c check_bg_jobs_reset`;
}
elseif($ARGV[0] eq "WARNING") #What to do in the WARNING State...
{
    system("NAGIOS PLUGINS DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\" \"$relativeLogPath $htmlFile");
}
elseif($ARGV[0] eq "UNKNOWN") #What to do in the UNKNOWN State...
{
}
elseif($ARGV[0] eq "CRITICAL") #What to do in the CRITICAL State...
{
    system("NAGIOS PLUGINS DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\" \"$relativeLogPath $htmlFile");
}
exit 0;
```

E.1.3 Other Check BG event handlers

We have included code for each of these event handlers:

- ▶ Check BG Link Card Event Handler
- ▶ Check BG Midplane Event Handler
- ▶ Check BG Node Card Event Handler
- ▶ Check BG Node Event Handler
- ▶ Check BG Processor Card Event Handler
- ▶ Check BG Service Action Event Handler
- ▶ Check BG Service Card Event Handler

For example, this describes the Check BG Link Card Event Handler (explanations for the others are similar). This event handler runs on a state change for the BG link card service. On a WARNING or CRITICAL state, it will call build_html.pl to build an HTML file that will help in viewing the plug-in's log file using Nagios.

This script is triggered when the BG link card service enters a new state. Conditionals that this script process by default are OK, WARNING, CRITICAL, and UNKNOWN. Upon a warning or critical state, the file build_html.pl will be called with the correct arguments and it will create an html file for viewing log files and making external commands on a service.

There are several hard-coded aspects to consider when implementing this script:

- ▶ Variables declared at the top of the file, such as pluginName, logName, htmlName, relativeLogPath, and htmlFile
- ▶ Path to the build_html.pl plug-in
- ▶ Any paths within the OK conditional
- ▶ Any arguments within the OK conditional

In reference to `"/usr/lib/nagios/plugins/check_nrpe -H "$ARGV[3]" -c check_db2_health_critical_reset";` The 'check_db2_health_critical_reset' needs to be changed if you modify the NRPE command file and rename the command.

Usage

To use:

ARGV[0]: Service State (\$SERVICESTATE\$ within Nagios)
ARGV[1]: Hostname (\$HOSTNAME\$ within Nagios)
ARGV[2]: Service Name (\$SERVICEDESC\$ within Nagios)
ARGV[3]: Host Address (\$HOSTADDRESS\$ within Nagios)

Return value

The return value is 0.

E.1.3.1 Check BG link card event handler code

Example E-3 shows the Check BG link card event handler code.

Example: E-3 Check BG link card event handler code

```
#!/usr/bin/perl
#####
# File Name: check_bg_linkcard_event.pl
#
# Author: Blue Gene Speed Team, Summer 2007
# Date: 7/24/2007
#
# Description: This event handler runs on a state change for the BG Linkcard
#service. On a WARNING or CRITICAL state, it will call build_html.pl to build an
#HTML file that will help in viewing the plug-in's log file using Nagios.
#
# Parameters:
# ARGV[0]: Service State ($SERVICESTATE$ within Nagios)
# ARGV[1]: Hostname ($HOSTNAME$ within Nagios)
# ARGV[2]: Service Name ($SERVICEDESC$ within Nagios)
# ARGV[3]: Host Address ($HOSTADDRESS$ within Nagios)
#
#####
# Other: This script is triggered when BG link card service enters a new state.
#Conditions that this script process by default are OK, WARNING, CRITICAL, and
#UNKNOWN. Upon a warning or critical state, the file build_html.pl will be called
#with the correct arguments and it will create an html file for viewing log files
#and making external commands on a service. #
#####
# Edit the below variables as needed for your plug-in.
my $pluginName = "check_bg_linkcard";           # ie 'check_ping'
my $logName = "linkcard.log";                   # ie criticalerror.log
my $htmlName = "linkcard.html";                 # ie criticalerror.html
my $relativeLogPath = "RELATIVE URL LOG PATH";  # Web path of the
log files
my $htmlFile="PHYSICAL LOG PATH";               # Log directory within the html
directory on your server
# Construct $relativeLogPath and $htmlFile using the IP address to delineate which
host the files belong too
```

```

$relativeLogPath = $relativeLogPath . $ARGV[3] . "/" . $pluginName . "/" .
$logName;
$htmlFile = $htmlFile . $ARGV[3] . "/" . $pluginName . "/" . $htmlName;
# What state is the service in?
if($ARGV[0] eq "OK") #What to do in the OK State...
{
}
elseif($ARGV[0] eq "WARNING") #What to do in the WARNING State...
{
    system("NAGIOS PLUGINS DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\" \"$relativeLogPath $htmlFile");
}
elseif($ARGV[0] eq "UNKNOWN") #What to do in the UNKNOWN State...
{
}
elseif($ARGV[0] eq "CRITICAL") #What to do in the CRITICAL State...
{
    system("NAGIOS PLUGINS DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\" \"$relativeLogPath $htmlFile");
}
exit 0;

```

E.1.3.2 Check BG midplane event handler code

Example E-4 shows the Check BG midplane event handler code.

Example: E-4 Check BG midplane event handler code

```

#!/usr/bin/perl
#####
# File Name: check_bg_midplane_event.pl
#
# Author: Blue Gene Speed Team, Summer 2007
# Date: 7/24/2007
#
# Description: This event handler runs on a state change for the BG Midplane
# service. On a WARNING or CRITICAL state, it will call build_html.pl to build an
# HTML file that will help in viewing the plug-in's log file using Nagios.
#
# Parameters:
# ARGV[0]: Service State ($SERVICESTATE$ within Nagios)
# ARGV[1]: Hostname ($HOSTNAME$ within Nagios)
# ARGV[2]: Service Name ($SERVICEDESC$ within Nagios)
# ARGV[3]: Host Address ($HOSTADDRESS$ within Nagios)
#
#####
# Other: This script is triggered when BG Midplane service enters a new state.
# Conditions that this script process by default are OK, WARNING, CRITICAL, and
# UNKNOWN. Upon a warning or critical state, the file build_html.pl will be called
# with the correct arguments and it will create an html file for viewing log files
# and making external commands on a service. #
#####
# Edit the below variables as needed for your plug-in.
my $pluginName = "check_bg_midplane";           # ie 'check_ping'
my $logName = "midplane.log";                   # ie criticalerror.log
my $htmlName = "midplane.html";                 # ie criticalerror.html

```

```

my $relativeLogPath = "RELATIVE URL LOG PATH";          # Web path of the
log files
my $htmlFile="PHYSICAL LOG PATH";          # Log directory within the html
directory on your server
# Construct $relativeLogPath and $htmlFile using the IP address to delineate which
host the files belong too
$relativeLogPath = $relativeLogPath . $ARGV[3] . "/" . $pluginName . "/" .
$logName;
$htmlFile = $htmlFile . $ARGV[3] . "/" . $pluginName . "/" . $htmlName;
# What state is the service in?
if($ARGV[0] eq "OK") #What to do in the OK State...
{
}
elsif($ARGV[0] eq "WARNING") #What to do in the WARNING State...
{
    system("NAGIOS PLUGINS DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\"
$relativeLogPath $htmlFile");
}
elsif($ARGV[0] eq "UNKNOWN") #What to do in the UNKNOWN State...
{
}
elsif($ARGV[0] eq "CRITICAL") #What to do in the CRITICAL State...
{
    system("NAGIOS PLUGINS DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\"
$relativeLogPath $htmlFile");
}
exit 0;

```

E.1.3.3 Check BG node card event handler code

Example E-5 shows the Check BG node card event handler code.

Example: E-5 Check BG node card event handler code

```

#!/usr/bin/perl
#####
# File Name: check_bg_midplane_event.pl
#
# Author: Blue Gene Speed Team, Summer 2007
# Date: 7/24/2007
#
# Description: This event handler runs on a state change for the BG Midplane
# service. On a WARNING or CRITICAL state, it will call build_html.pl to build an
# HTML file that will help in viewing the plug-in's log file using Nagios.
#
# Parameters:
# ARGV[0]: Service State ($SERVICESTATE$ within Nagios)
# ARGV[1]: Hostname ($HOSTNAME$ within Nagios)
# ARGV[2]: Service Name ($SERVICEDESC$ within Nagios)
# ARGV[3]: Host Address ($HOSTADDRESS$ within Nagios)
#
#####
# Other: This script is triggered when BG Midplane service enters a new state.
#Conditions that this script process by default are OK, WARNING, CRITICAL, and
#UNKNOWN. Upon a warning or critical state, the file build_html.pl will be called

```



```

#with the correct arguments and it will create an html file for viewing log files
#and making external commands on a service. #
#####
# Edit the below variables as needed for your plug-in.
my $pluginName = "check_bg_midplane";           # ie 'check_ping'
my $logName = "midplane.log";                   # ie criticalerror.log
my $htmlName = "midplane.html";                 # ie criticalerror.html
my $relativeLogPath = "RELATIVE URL LOG PATH";  # Web path of the
log files
my $htmlFile="PHYSICAL LOG PATH";               # Log directory within the html
directory on your server
# Construct $relativeLogPath and $htmlFile using the IP address to delineate which
host the files belong too
$relativeLogPath = $relativeLogPath . $ARGV[3] . "/" . $pluginName . "/" .
$logName;
$htmlFile = $htmlFile . $ARGV[3] . "/" . $pluginName . "/" . $htmlName;
# What state is the service in?
if($ARGV[0] eq "OK") #What to do in the OK State...
{
}
elsif($ARGV[0] eq "WARNING") #What to do in the WARNING State...
{
    system("NAGIOS_PLUGINS_DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\"
$relativeLogPath $htmlFile");
}
elsif($ARGV[0] eq "UNKNOWN") #What to do in the UNKNOWN State...
{
}
elsif($ARGV[0] eq "CRITICAL") #What to do in the CRITICAL State...
{
    system("NAGIOS_PLUGINS_DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\"
$relativeLogPath $htmlFile");
}
exit 0;

```

E.1.3.4 Check BG node event handler code

Example E-6 shows the Check BG node event handler code.

Example: E-6 Check BG node event handler code

```

#!/usr/bin/perl
#####
# File Name: check_bg_node_event.pl
#
# Author: Blue Gene Speed Team, Summer 2007
# Date: 7/24/2007
#
# Description: This event handler runs on a state change for the BG Node service.
#On a WARNING or CRITICAL state, it will call build_html.pl to build an HTML file
#that will help in viewing the plug-in's log file using Nagios.
#
# Parameters:
# ARGV[0]: Service State ($SERVICESTATE$ within Nagios)
# ARGV[1]: Hostname ($HOSTNAME$ within Nagios)
# ARGV[2]: Service Name ($SERVICEDESC$ within Nagios)

```

```

# ARGV[3]: Host Address ($HOSTADDRESS$ within Nagios)
#
#####
# Other: This script is triggered when BG Node service enters a new state.
#Conditions that this script process by default are OK, WARNING, CRITICAL, and
#UNKNOWN. Upon a warning or critical state, the file build_html.pl will be called
#with the correct arguments and it will create an html file for viewing log files
#and making external commands on a service.
#####
# Edit the below variables as needed for your plug-in.
my $pluginName = "check_bg_node";          # ie 'check_ping'
my $logName = "node.log";                  # ie criticalerror.log
my $htmlName = "node.html";                # ie criticalerror.html
my $relativeLogPath = "RELATIVE URL LOG PATH";      # Web path of the
log files
my $htmlFile="PHYSICAL LOG PATH";          # Log directory within the html
directory on your server
# Construct $relativeLogPath and $htmlFile using the IP address to delineate which
host the files belong too
$relativeLogPath = $relativeLogPath . $ARGV[3] . "/" . $pluginName . "/" .
$logName;
$htmlFile = $htmlFile . $ARGV[3] . "/" . $pluginName . "/" . $htmlName;
# What state is the service in?
if($ARGV[0] eq "OK") #What to do in the OK State...
{
}
elsif($ARGV[0] eq "WARNING") #What to do in the WARNING State...
{
    system("NAGIOS_PLUGINS_DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\"
$relativeLogPath $htmlFile");
}
elsif($ARGV[0] eq "UNKNOWN") #What to do in the UNKNOWN State...
{
}
elsif($ARGV[0] eq "CRITICAL") #What to do in the CRITICAL State...
{
    system("NAGIOS_PLUGINS_DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\"
$relativeLogPath $htmlFile");
}
exit 0;

```

E.1.3.5 Check BG processor card event handler code

Example E-7 is the Check BG processor card event handler code.

Example: E-7 Check BG processor card event handler code

```

#!/usr/bin/perl
#####
# File Name: check_bg_processorcard_event.pl
#
# Author: Blue Gene Speed Team, Summer 2007
# Date: 7/24/2007
#
# Description: This event handler runs on a state change for the BG Processorcard
#service. On a WARNING or CRITICAL state, it will call build_html.pl to build an

```

```

#HTML file that will help in viewing the plug-in's log file using Nagios.
#
# Parameters:
# ARGV[0]: Service State ($SERVICESTATE$ within Nagios)
# ARGV[1]: Hostname ($HOSTNAME$ within Nagios)
# ARGV[2]: Service Name ($SERVICEDESC$ within Nagios)
# ARGV[3]: Host Address ($HOSTADDRESS$ within Nagios)
#####
# Other: This script is triggered when BG Processorcard service enters a new
#state. Conditions that this script process by default are OK, WARNING, CRITICAL,
#and UNKNOWN. Upon a warning or critical state, the file build_html.pl will be
#called with the correct arguments and it will create an html file for viewing log
#files and making external commands on a service. #
#####
# Edit the below variables as needed for your plug-in.
my $pluginName = "check_bg_processorcard";           # ie 'check_ping'
my $logName = "processorcard.log";                   # ie criticalerror.log
my $htmlName = "processorcard.html";                 # ie criticalerror.html
my $relativeLogPath = "RELATIVE URL LOG PATH";       # Web path of the
log files
my $htmlFile="PHYSICAL LOG PATH";                    # Log directory within the html
directory on your server
# Construct $relativeLogPath and $htmlFile using the IP address to delineate which
host the files belong too
$relativeLogPath = $relativeLogPath . $ARGV[3] . "/" . $pluginName . "/" .
$logName;
$htmlFile = $htmlFile . $ARGV[3] . "/" . $pluginName . "/" . $htmlName;
# What state is the service in?
if($ARGV[0] eq "OK") #What to do in the OK State...
{
}
elseif($ARGV[0] eq "WARNING") #What to do in the WARNING State...
{
    system("NAGIOS_PLUGINS_DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\"
$relativeLogPath $htmlFile");
}
elseif($ARGV[0] eq "UNKNOWN") #What to do in the UNKNOWN State...
{
}
elseif($ARGV[0] eq "CRITICAL") #What to do in the CRITICAL State...
{
    system("NAGIOS_PLUGINS_DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\"
$relativeLogPath $htmlFile");
}
exit 0;

```

E.1.3.6 Check BG service card event handler code

Example E-8 shows the Check BG service card event handler code.

Example: E-8 Check BG service card event handler

```

#!/usr/bin/perl
#####
# File Name: check_bg_servicecard_event.pl
#

```

```

# Author: Blue Gene Speed Team, Summer 2007
# Date: 7/24/2007
#
# Description: This event handler runs on a state change for the BG Servicecard
#service. On a WARNING or CRITICAL state, it will call build_html.pl to build an
#HTML file that will help in viewing the plug-in's log file using Nagios.
#
# Parameters:
# ARGV[0]: Service State ($SERVICESTATE$ within Nagios)
# ARGV[1]: Hostname ($HOSTNAME$ within Nagios)
# ARGV[2]: Service Name ($SERVICEDESC$ within Nagios)
# ARGV[3]: Host Address ($HOSTADDRESS$ within Nagios)
#####
# Other: This script is triggered when BG Servicecard service enters a new state.
#Conditions that this script process by default are OK, WARNING, CRITICAL, and
#UNKNOWN. Upon a warning or critical state, the file build_html.pl will be called
#with the correct arguments and it will create an html file for viewing log files
#and making external commands on a service. #
#####
# Edit the below variables as needed for your plug-in.
my $pluginName = "check_bg_servicecard";           # ie 'check_ping'
my $logName = "servicecard.log";                  # ie criticalerror.log
my $htmlName = "servicecard.html";                # ie criticalerror.html
my $relativeLogPath = "RELATIVE URL LOG PATH";    # Web path of the
log files
my $htmlFile="PHYSICAL LOG PATH";                 # Log directory within the html
directory on your server
# Construct $relativeLogPath and $htmlFile using the IP address to delineate which
host the files belong too
$relativeLogPath = $relativeLogPath . $ARGV[3] . "/" . $pluginName . "/" .
$logName;
$htmlFile = $htmlFile . $ARGV[3] . "/" . $pluginName . "/" . $htmlName;
# What state is the service in?
if($ARGV[0] eq "OK") #What to do in the OK State...
{
}
elseif($ARGV[0] eq "WARNING") #What to do in the WARNING State...
{
    system("NAGIOS_PLUGINS_DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\"
$relativeLogPath $htmlFile");
}
elseif($ARGV[0] eq "UNKNOWN") #What to do in the UNKNOWN State...
{
}
elseif($ARGV[0] eq "CRITICAL") #What to do in the CRITICAL State...
{
    system("NAGIOS_PLUGINS_DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\"
$relativeLogPath $htmlFile");
}
exit 0;

```

E.1.3.7 Check BG service action event handler code

Example E-9 is the Check BG service action event handler.

Example: E-9 Check BG service action event handler code

```
#!/usr/bin/perl
#####
# File Name: check_bg_serviceaction_event.pl
#
# Author: Blue Gene Speed Team, Summer 2007
# Date: 7/24/2007
#
# Description: This event handler runs on a state change for the BG Service Action
#service. On a WARNING or CRITICAL state, it will call build_html.pl to build an
#HTML file that will help in viewing the plug-in's log file using Nagios.
#
# Parameters:
#
# ARGV[0]: Service State ($SERVICESTATE$ within Nagios)
#
# ARGV[1]: Hostname ($HOSTNAME$ within Nagios)
#
# ARGV[2]: Service Name ($SERVICEDESC$ within Nagios)
#
# ARGV[3]: Host Address ($HOSTADDRESS$ within Nagios)
#
#####
# Other: This script is triggered when BG Service Action service enters a new
#state. Conditions that this script process by default are OK, WARNING, CRITICAL,
#and UNKNOWN. Upon a warning or critical state, the file build_html.pl will be
#called with the correct arguments and it will create an html file for viewing log
#files and making external commands on a service. #
#####
# Edit the below variables as needed for your plug-in.
my $pluginName = "check_bg_serviceaction";          # ie 'check_ping'
my $logName = "serviceaction.log";                  # ie criticalerror.log
my $htmlName = "serviceaction.html";                # ie criticalerror.html
my $relativeLogPath = "RELATIVE URL LOG PATH";      # Web path of the
log files
my $htmlFile="PHYSICAL LOG PATH";                   # Log directory within the html
directory on your server
# Construct $relativeLogPath and $htmlFile using the IP address to delineate which
host the files belong too
$relativeLogPath = $relativeLogPath . $ARGV[3] . "/" . $pluginName . "/" .
$logName;
$htmlFile = $htmlFile . $ARGV[3] . "/" . $pluginName . "/" . $htmlName;
# What state is the service in?
if($ARGV[0] eq "OK") #What to do in the OK State...
{
}
elseif($ARGV[0] eq "WARNING") #What to do in the WARNING State...
{
    system("NAGIOS_PLUGINS_DIRECTORY/build_html.pl \"\$ARGV[1]\" \"\$ARGV[2]\" \"
$relativeLogPath $htmlFile");
}
elseif($ARGV[0] eq "UNKNOWN") #What to do in the UNKNOWN State...
```

```

{
}
elseif($ARGV[0] eq "CRITICAL") #What to do in the CRITICAL State...
{
    system("NAGIOS_PLUGINS_DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\" \"$relativeLogPath $htmlFile");
}
exit 0;

```

E.1.4 Check DB2 Health Event Handler

There are two types:

- ▶ Check DB2 Health Critical Event Handler
- ▶ Check DB2 Health Warning Event Handler

For example, the Check DB2 Health Critical Event Handler is for the DB2 Health Critical service (check_db2_health.pl --alertlevel=critical). It just deals with the OK and CRITICAL states.

This script is triggered when the DB2 Health Critical service enters a new state. Upon a critical state, the file build_html.pl will be called with the correct arguments and it will create an html file for viewing and clearing the DB2 Health Critical service messages. If the service transitions into an OK state, this script will call a script on the host with an error that will remove the log file and reset the state of the service.

There are several hard-coded considerations when implementing this script:

- ▶ Variables declared at the top of the file, such as pluginName, logName, htmlName, relativeLogPath, and htmlFile
- ▶ Path to the build_html.pl plug-in
- ▶ Any paths within the OK conditional
- ▶ Any arguments within the OK conditional

In reference to `"/usr/lib/nagios/plugins/check_nrpe -H "$ARGV[3]" -c check_db2_health_critical_reset";` The 'check_db2_health_critical_reset' needs to be changed if you modify the NRPE command file and rename the command.

Usage

To use:

```

ARGV[0]: Service State ($SERVICESTATE$ within Nagios)
ARGV[1]: Hostname ($HOSTNAME$ within Nagios)
ARGV[2]: Service Name ($SERVICEDESC$ within Nagios)
ARGV[3]: Host Address ($HOSTADDRESS$ within Nagios)

```

Return value

The return value is 0.

E.1.4.1 Check DB2 health critical event handler code

Example E-10 is the Check DB2 health critical event handler.

Example: E-10 Check DB2 health critical event handler code

```
#!/usr/bin/perl
#####
# File Name: check_db2_health_critical_event.pl
# Author: Blue Gene Speed Team
# Date: Summer, 2007
# Description: This event handler is for the DB2 Health Critical service
# (check_db2_health.pl --alertlevel=critical). It just deals with the OK and
# CRITICAL states.
# Parameters:
# ARGV[0]: Service State ($SERVICESTATE$ within Nagios)
# ARGV[1]: Hostname ($HOSTNAME$ within Nagios)
# ARGV[2]: Service Name ($SERVICEDESC$ within Nagios)
# ARGV[3]: Host Address ($HOSTADDRESS$ within Nagios)
#####
# Other: This script is triggered when the DB2 Health Critical service enters a
#new state. Upon a critical state, the file build_html.pl will be called with the
#correct arguments and it will create an html file for viewing and clearing the
#DB2 Health Critical service messages. If the service transitions into an OK
#state, then this script will call a script on the erroring host that will remove
#the log file and reset the state of the service.
#####
# Edit the below variables as needed for your plug-in.
my $pluginName = "check_db2_health";          # ie 'check_ping'
my $logName = "critical.log";                 # ie criticalerror.log
my $htmlName = "critical.html";               # ie criticalerror.html
my $relativeLogPath = "RELATIVE URL LOG PATH"; # Web path of the
log files
my $htmlFile="PHYSICAL LOG PATH";             # Log directory within the html
directory on your server
# Construct $relativeLogPath and $htmlFile using the IP address to delineate which
host the files belong too
$relativeLogPath = $relativeLogPath . $ARGV[3] . "/" . $pluginName . "/" .
$logName;
$htmlFile = $htmlFile . $ARGV[3] . "/" . $pluginName . "/" . $htmlName;
# What state is the service in?
if($ARGV[0] eq "OK") #What to do in the OK State...
{
    `NAGIOS PLUGINS DIRECTORY/check_nrpe -H "$ARGV[3]" -c
check_db2_health_critical_reset`;
}
elseif($ARGV[0] eq "WARNING") #What to do in the WARNING State...
{
}
elseif($ARGV[0] eq "UNKNOWN") #What to do in the UNKNOWN State...
{
}
elseif($ARGV[0] eq "CRITICAL") #What to do in the CRITICAL State...
{
    system("NAGIOS PLUGINS DIRECTORY/build_html.pl \"${ARGV[1]}\" \"${ARGV[2]}\"
$relativeLogPath $htmlFile");
}
```

```
exit 0;
```

E.1.4.2 Check DB2 health warning event handler code

Example E-11 is the Check DB2 health warning event handler code.

Example: E-11 Check DB2 health warning event handler code

```
#!/usr/bin/perl
#####
# File Name: check_db2_health_warning_event.pl
# Author: Blue Gene Speed Team
# Date: Summer, 2007
# Description: This event handler is for the DB2 Health Warning service
# (check_db2_health.pl --alertlevel=warning). It just deals with the OK and
# WARNING states.
# Parameters:
# ARGV[0]: Service State ($SERVICESTATE$ within Nagios)
# ARGV[1]: Hostname ($HOSTNAME$ within Nagios)
# ARGV[2]: Service Name ($SERVICEDESC$ within Nagios)
# ARGV[3]: Host Address ($HOSTADDRESS$ within Nagios)
#####
# Other: This script is triggered when the DB2 Health Warning service enters a new
#state. Upon a warning state, the file build_html.pl will be called with the
#correct arguments and it will create an html file for viewing and clearing the
#DB2 Health Warning service messages. If the service transitions into an OK state,
#then this script will call a script on the erroring host that will remove the log
#file and reset the state of the service.
#####
# Edit the below variables as needed for your plug-in.
my $pluginName = "check_db2_health";          # ie 'check_ping'
my $logName = "warning.log";                  # ie criticalerror.log
my $htmlName = "warning.html";                # ie criticalerror.html
my $relativeLogPath = "RELATIVE URL LOG PATH"; # Web path of the
log files
my $htmlFile="PHYSICAL LOG PATH";             # Log directory within the html
directory on your server
# Construct $relativeLogPath and $htmlFile using the IP address to delineate which
host the files belong too
$relativeLogPath = $relativeLogPath . $ARGV[3] . "/" . $pluginName . "/" .
$logName;
$htmlFile = $htmlFile . $ARGV[3] . "/" . $pluginName . "/" . $htmlName;
# What state is the service in?
if($ARGV[0] eq "OK") #What to do in the OK State...
{
    `NAGIOS PLUGINS DIRECTORY/check_nrpe -H "$ARGV[3]" -c
check_db2_health_warning_reset`;
}
elseif($ARGV[0] eq "WARNING") #What to do in the WARNING State...
{
    system("NAGIOS PLUGINS DIRECTORY/build_html.pl \"$ARGV[1]\" \"$ARGV[2]\"
$relativeLogPath $htmlFile");
}
elseif($ARGV[0] eq "UNKNOWN") #What to do in the UNKNOWN State...
{
}
}
```



```

elseif($ARGV[0] eq "CRITICAL") #What to do in the CRITICAL State...
{
}
exit 0;

```

E.2 Build HTML

This plug-in builds an HTML file for use with Nagios. This HTML file displays a text box and a Submit button for sending an external “OK” command to the service you specify. It also displays a log file in an iframe beneath the text box.

There are several hard-coded aspects to consider when implementing this script:

- ▶ Any hard-coded paths within the file (stylesheets, cmd.cgi, and so forth if there are any more)
- ▶ Make sure the nagios/nagcmd is the correct user/group in the `chown` command at the end of the file.

Usage

To use:

```

./build_html.pl <$HOSTNAME$> <$SERVICENAME$> <LOG_FILE> <HTML_FILE>
<$HOSTNAME$> - Hostname of the host as described in Nagios
<$SERVICENAME$> - Servicename as described in Nagios
<LOG_FILE> - Internet path to log file (that is, "/nagios/logs/logfile.log")
<HTML_FILE> - Full path to HTML file (that is, "/srv/htdocs/htmlfile.html")

```

Return value

The return value is 0.

Code

Example E-12 is the Build HTML.

Example: E-12 Build HTML

```

#!/usr/bin/perl
#####
# File Name: build_html.pl
# Author: Blue Gene Speed Team, Summer 2007
# Date: 7/25/2007
# Description: This plug-in builds an HTML file for use with Nagios. This HTML
file
#displays a text box and submit button for sending an external "OK" command to the
#service you specify. It also displays a log file in an iframe below the text box.
#
# Usage: ./build_html.pl <$HOSTNAME$> <$SERVICENAME$> <LOG_FILE> <HTML_FILE>
#      <$HOSTNAME$> - Hostname of the host as described in Nagios
#      <$SERVICENAME$> - Servicename as described in Nagios
#      <LOG_FILE> - Internet path to log file (e.g. "/nagios/logs/logfile.log")
#      <HTML_FILE> - Full path to HTML file (e.g. "/srv/htdocs/htmlfile.html")
# Return Value: 0
#####
# Other:

```

```
#####
my $hostname = $ARGV[0];
my $servicename = $ARGV[1];
my $logfile = $ARGV[2];
my $htmlfile = $ARGV[3];
open(HTMLFILE, ">${htmlfile}") or die("ERROR\n");
print HTMLFILE << "EOF";
<html>
<head><title>External Command Interface</title>
<link rel="stylesheet" type="text/css" href="/nagios/stylesheets/common.css">
<link rel="stylesheet" type="text/css" href="/nagios/stylesheets/cmd.css"></head>
<body class="cmd">
<form method='POST' action='/nagios/cgi-bin/cmd.cgi'>
<INPUT TYPE='HIDDEN' NAME='cmd_typ' VALUE='30'>
<input name="cmd_mod" value="2" type="hidden">
<input name="host" value="${hostname}" type="hidden">
<input name="service" value="${servicename}" type="hidden">
<input name="plugin_state" value="0" type="hidden">
<table class="optBox" cellpadding="0" cellspacing="0">
<tr><td class="optBoxItem"><b>Clear Errors</b></td></tr>
<tr><td class="optBoxItem">Status Information:</td><td><b><input
name="plugin_output" value="" type="text"></b></td>
<td class="optBoxItem"></td><td class="optBoxItem"><input name="btnSubmit"
value="Submit" type="submit"></td></tr>
</table>
</form>
<hr>
<h3>Log File:</h3>
<iframe style="background-color: white;" src = "${logfile}" width="100%"
height="75%">
</iframe>
</body>
</html>
EOF
close(HTMLFILE);
system("sudo chown NAGIOS_USER.NAGIOS_GROUP ${htmlfile}");
system("sudo chmod 775 ${htmlfile}");
```

E.3 Check BG jobs reset

This script clears the log file for `check_bgl_` or `check_bgp_jobs.pl` and sets the state files so that any error after this point will be recorded, but none before. This script is meant to be called from an event handler by Nagios. That way, when BG Jobs is set to an OK state, this script is called, clearing the log file and setting the state files correctly.

There are several hard-coded considerations when implementing this script:

- ▶ Make sure the path to the state files is correct.
- ▶ Make sure the path to the log file is correct.

Usage

To use:

```
./check_bg_jobs_reset.sh
```

Return value

The return value is 0.

Code

Example E-13 is code for the Check BG jobs reset.

Example: E-13 Check BG jobs reset

```
#!/bin/sh
#####
# File Name: check_bg_jobs_reset.sh
# Author: Blue Gene Speed Team
# Date: Summer, 2007
# Description: Clears the log file for check_bg1_ or check_bgp_jobs.pl and sets
#the state files such that any error after this point will be recorded, but none
#before. This script is meant to be called from an event handler by Nagios. That
#way, when BG Jobs is set to an OK state, this script is called, clearing the log
#file, and setting the state files correctly.
# Usage: ./check_bg_jobs_reset.sh
# Return Value: 0
#####
# Other:
#####
cp STATE_FILE_DIRECTORY_HERE/jobid.error STATE_FILE_DIRECTORY_HERE/jobid.clear
> LOG_FILE_DIRECTORY_HERE/joberrors.log
```

E.4 Check DB2 health reset

This script resets the state and log files for the check_db2_health.pl plug-in.

There are several variables towards the top of the script that have been hard-coded. These include IP addresses and paths to file locations. Change these variables to match the environment:

currentWarningStateFile The local path to the warning state file used by check_db2_health.pl

currentCriticalStateFile The local path to the critical state file used by check_db2_health.pl

warningLog The local path to the warning log file used by check_db2_health.pl

criticalLog The local path to the criticallog file used by check_db2_health.pl

Usage

To use:

```
./check_db2_health_reset.pl --alertlevel=[CRITICAL|WARNING]
```

Return value

The return value is 0.

Code

Example E-14 on page 122 is the Check DB2 health reset.

Example: E-14 Check DB2 health reset

```
#!/usr/bin/perl
#####
# File Name: check_db2_health_reset.pl
# Author: Blue Gene Speed Team
# Date: Summer, 2007
# Description: This script resets the state and log files for the
#check_db2_health.pl plug-in
# Usage: ./check_db2_health_reset.pl --alertlevel=[CRITICAL|WARNING]
# Return Value: None
#####
# Other:
#####
use Getopt::Long;
# Declare variables
my $alertlevel;
my $warningStateFile = "STATE_FILE_DIRECTORY_HERE/warning.STATE";
my $criticalStateFile = "STATE_FILE_DIRECTORY_HERE/critical.STATE";
my $warningLog = "LOG_FILE_DIRECTORY_HERE/warning.log";
my $criticalLog = "LOG_FILE_DIRECTORY_HERE/critical.log";
# Store command line alertlevel input to var
$result = GetOptions ("alertlevel=s" => \$alertlevel);
# Convert the alertlevel argument to all uppercase.
$alertlevel = uc($alertlevel);
# Check to make sure correct argument was specified (--alertlevel
[WARNING|CRITICAL])
if (($alertlevel ne "WARNING") && ($alertlevel ne "CRITICAL") && ($sysdbuser eq
""))
{
    # Invalid arguments specified, exit.
    print "Invalid parameters.\n";
    print "Correct Usage: ./check_db2_health_reset.pl
--alertlevel=[WARNING|CRITICAL]\n";
    exit 1;
}
# Reset db2 health log and state files for proper plug-ins
if ($alertlevel eq "WARNING")
{
    # Delete the log file
    unlink(${warningLog});
    # Write OK to state file
    open CURRENTSTATE, '>', ${warningStateFile} or die "Can't open
${warningStateFile} $!";
    print CURRENTSTATE "OK";
    close(CURRENTSTATE);
}
else
{
    # Delete the log file
    unlink(${criticalLog});
    # Write OK to state file
    open CURRENTSTATE, '>', ${criticalStateFile} or die "Can't open
${criticalStateFile} $!";
    print CURRENTSTATE "OK";
    close(CURRENTSTATE);
}
```

```
}  
exit 0;
```

Archived

Archived



NRPE scripts

This appendix contains NRPE scripts.

F.1 Install NRPE

This is a generalized template of a script created to install NRPE on destination hosts. It works by compressing the current NRPE master folder installed on bg-management and distributing it to all hosts and performing the necessary installation steps.

However, this script is very environment-specific, so it has been reduced mostly to reading in specified hosts from a hostips file and then listing the logical steps to install NRPE on your environment. The steps will probably differ somewhat so make changes where necessary.

Also, change anything listed in capitalized letters (uppercase), any paths to files, anything listed with the text of "MODIFY:", and anything else where necessary.

Code

Example F-1 is the code to install NRPE.

Example: F-1 Install NRPE

```
#!/usr/bin/perl
#####
# File Name: installNRPE.pl
# Author: Blue Gene Speed Team
# Date: Summer, 2007
#####
# Description - This script is used to install NRPE on one or more host machines.
#The script tars up the current nrpe setup located in the nrpe master directory.
#It then copies the tar file to all hosts specified in the 'hostips' file. The
#script is then untarred and NRPE is set up on the remote host. After installation
#completes, both the remote host and the localhost are cleaned of any unneeded
#installation files/directories.
#
# Usage - All hosts that you wish to install NRPE on, must be entered in a file
#with the path# hostips. Each host must be listed on a separate line.
#####
my $ipFile = "PATH_TO_HOSTIPS_FILE";
my $ipAddresses;
my $destHost;
my $arch;
print "Reading file \"${ipFile}\" for the list of destination hosts...\n";
# Read in IP addresses to install the NRPE daemon
open(DAT, $ipFile) || die("Could not open \"${ipFile}\" file. Please list the IPs
you wish to manage in a file \"${ipFile}\"");
@fileData=<DAT>;
chomp(@fileData);
close(DAT);
# Strip out any blank lines and comments (#)
@ipAddresses = grep { $_ ne "" } @fileData;
@ipAddresses = grep { m/^[^#]/ } @ipAddresses;
# Print each server inputed
foreach(@ipAddresses) {
    print "$_\n";
}
# Display reinstall warning
print("\n#####");
```



```

print("\nWARNING: Running the install script on a host that already has NRPE
installed will fully reinstall NRPE and reset all plugins to their default
reinstall state.\nIf you wish to reinstall everything but the NRPE state and log
files, please run the reinstallNRPE.pl script instead.");
print("\n#####\n");
# Make the user confirm the installation process for the servers. Continue looping
until yes/no/y/n is entered.
do {
    print "\nInstall the NRPE user, group, and daemon on the above servers?
<y|n>:";
    $input = <STDIN>;
    chomp($input);
} while (($input ne "yes") && ($input ne "no") && ($input ne "y") && ($input ne
"n"));
# If the user enters no|n, exit.
if (($input eq "no") || ($input eq "n")) {
    exit 1;
}
print "-----\n";
# Compress NRPE installation to send
print "Compressing NRPE install files in NRPEMASTERFOLDER...";
system("tar -f /tmp/nrpe.tgz -C NRPE_MASTER_FOLDER -cz nrpe 1> /dev/null");
print "DONE\n";
print "-----\n\n\n";
# Setup the NRPE daemon on each host listed
foreach(@ipAddresses) {
    # Set destHost as installation host
    $destHost = $_;
    # Stop the NRPE daemon whether or not it is installed and running.
    print "Stopping any existing NRPE daemons...";
    # MODIFY: SSH to remote host and stop running NRPE daemon
    # Setup group/user on host
    print "Setting up group GROUPHERE and member (user) USERHERE...";
    # MODIFY: SSH to remote host and set up all necessary user/groups
    print "DONE\n";
    # Make appropriate directories
    print "Creating directories for installation...";
    # MODIFY: SSH to remote host and create install directories
    print "DONE\n";
    # Copy over NRPE related files
    print "Copying NRPE compressed archive and uncompressing...";
    # MODIFY: SCP over NRPE tar file and SSH and untar
    print "DONE\n";
    # Installing NRPE in correct locations
    print "Installing NRPE...";
    # MODIFY: SSH to remote host and install NRPE in correct locations
    print "DONE\n";
    # Link init scripts to proper locations
    print "Setting up init scripts...";
    # MODIFY: SSH to remote host and link init scripts to proper locations
    print "DONE\n";
    # Set Permissions
    print "Setting up permissions...";
    # MODIFY: SSH to remote host and set permissions for all installed files.
    print "DONE\n";
}

```

```

        # Clean up dest host
        print "Cleaning up host installation files...";
        # MODIFY: SSH to remote host and remove all installation related
directories
        print "DONE\n";
        # Start NRPE Daemon
        print "Starting NRPE daemon...";
        # MODIFY: Command to ssh to remote host and start the nrpe daemon
        # MODIFY: system("");
        print "...installation completed on ${destHost}.\n\n";
    }
    print "\n\n-----\n";
    # Clean up localhost
    print "Cleaning up localhost file...";
    system("rm /tmp/nrpe.tgz 1> /dev/null");
    print "DONE\n";
    print "Installation finished successfully.\n";
    exit 0;

```

F.2 Reinstall NRPE

This is a generalized template of a script created to reinstall NRPE on destination hosts. It works by compressing the current NRPE master folder installed on bg-management, distributing it to all hosts, and performing the necessary installation steps. It works the same way as the install script, but does not recreate the user/group and does not copy files over that have to do with plug-in states.

However, this script is very environment-specific, so it has been reduced mostly to reading in specified hosts from a hostips file and then listing the logical steps to install NRPE on your environment. The steps will probably differ somewhat, so make changes where necessary.

Also, change anything listed in capitalized letters (uppercase), any paths to files, anything listed with the text of “MODIFY: “, and anything else where necessary.

Code

Example F-2 is the code to reinstall NRPE.

Example: F-2 Reinstall NRPE

```

#!/usr/bin/perl
#####
# File Name: reinstallNRPE.pl
# Author: Blue Gene Speed Team
# Date: Summer 2007
#####
# Description - This script is used to reinstall NRPE on one or more host
#machines. The script has the same functionality of installNRPE.pl, but does not
#add the group/user or the directories that contain the state files. This allows
#the NRPE installation to be updated while retaining the current state of the
#plug-ins before the reinstall.      #
#
#Usage - All hosts that you wish to reinstall NRPE on must be entered in a file
#with the path to a file named hostips. Each host must be listed on a separate

```

```

#line.
#####
my $ipFile = "PATH_TO_HOSTIPS_FILE";
my $ipAddresses;
my $destHost;
my $arch;
print "Reading file \"${ipFile}\" for the list of destination hosts...\n";
# Read in IP addresses to install the NRPE daemon
open(DAT, $ipFile) || die("Could not open \"${ipFile}\" file. Please list the IPs
you wish to manage in a file \"${ipFile}\"");
@fileData=<DAT>;
chomp(@fileData);
close(DAT);
# Strip out any blank lines and comments (#)
@ipAddresses = grep { $_ ne "" } @fileData;
@ipAddresses = grep { m/^[^#]/ } @ipAddresses;
# Print each server inputed
foreach(@ipAddresses) {
    print "$_\n";
}
# Make the user confirm the installation process for the servers. Continue looping
until yes/no/y/n is entered.
do {
    print "\nReinstall NRPE on the above servers? <y|n>:";
    $input = <STDIN>;
    chomp($input);
} while (($input ne "yes") && ($input ne "no") && ($input ne "y") && ($input ne
"n"));
# If the user enters no|n, exit.
if (($input eq "no") || ($input eq "n")) {
    exit 1;
}
print "-----\n";
# Compress NRPE installation to send
print "Compressing NRPE install files in NRPEMASTERDIR...";
# MODIFY: tar files up in nrpe master dir
print "DONE\n";
print "-----\n\n";
# Setup the NRPE daemon on each host listed
foreach(@ipAddresses) {
    # Set destHost as installation host
    $destHost = $_;
    # Stop the NRPE daemon whether or not it is installed and running.
    print "Stopping any existing NRPE daemons...";
    # MODIFY: SSH and stop daemon
    # Make appropriate directories
    print "Creating directories for installation...";
    # MODIFY: SSH and create install dirs
    print "DONE\n";
    # Copy over NRPE related files
    print "Copying NRPE compressed archive and uncompressing...";
    # MODIFY: SCP tar file and SSH and uncompress
    print "DONE\n";
    # Installing NRPE in correct locations
    print "Reinstalling NRPE...";
}

```

```

        # MODIFY: SSH and reinstall files that are NOT related to the state of
#plug-ins
    print "DONE\n";
    # Link init scripts to proper locations
    print "Setting up init scripts...";
    # MODIFY: SSH and setup init scripts
    print "DONE\n";
    # Set Permissions
    print "Setting up permissions...";
    # MODIFY: SSH and setup permissions
    print "DONE\n";
    # Clean up dest host
    print "Cleaning up host reinstallation files...";
    # MODIFY: clean up host install files
    print "DONE\n";
    # Start NRPE Daemon
    print "Starting NRPE daemon...";
    # MODIFY: SSH and start the NRPE daemon
    print "...reinstallation completed on ${destHost}.\n\n";
}
print "\n\n-----\n";
# Clean up localhost
print "Cleaning up localhost file...";
system("rm /tmp/nrpe.tgz 1> /dev/null");
print "DONE\n";
print "#####\n";
print "WARNING: No files related to plugin states were installed, and the
group/user were not added. This is because a reinstall only copies over the
binaries, scripts, and configuration files. Any plugin state files or log files
were not copied in order to save the current NRPE state on the end hosts.\n\n";
print "#####\n";
print "Reinstallation finished successfully.\n";
exit 0;

```

F.3 Remove NRPE

This is a generalized template of a script created to remove the NRPE user/group and NRPE itself on destination hosts. Change anything listed in CAPITAL letters (uppercase), any paths to files, anything listed with the text of “MODIFY: “, and anything else where necessary.

Code

Example F-3 shows how to remove NRPE.

Example: F-3 Remove NRPE

```

#!/usr/bin/perl
#####
# File Name: removeNRPE.pl
# Author: Blue Gene Speed Team
# Date: Summer 2007
#####

```

```

#Description - This script is used to uninstall NRPE from one or more host
#machines. The nagios/nagcmd user/group are removed, and any related files are
#also removed.
# Usage - All hosts that you wish to remove NRPE from, must be entered in a file
#with the path to a file named hostips. Each host must be listed on a separate
#line.
#####
my $ipFile = "PATH_TO_HOSTIPS_FILE";
my $ipAddresses;
my $destHost;
print "Reading file \"${ipFile}\" for the list of destination hosts...\n";
# Read in IP addresses to remove the NRPE daemon
open(DAT, $ipFile) || die("Could not open \"${ipFile}\" file. Please list the IPs
you wish to manage in a file \"${ipFile}\"");
@fileData=<DAT>;
chomp(@fileData);
close(DAT);
# Strip out any blank lines and comments (#)
@ipAddresses = grep { $_ ne "" } @fileData;
@ipAddresses = grep { m/^[^#]/ } @ipAddresses;
# Print each server inputed
foreach(@ipAddresses) {
    print "$_\n";
}
# Make the user confirm the remove process for the servers. Continue looping until
yes/no/y/n is entered.
do {
    print "\nRemove the NRPE user, group, and daemon on the above servers?
<y|n>:";
    $input = <STDIN>;
    chomp($input);
} while (($input ne "yes") && ($input ne "no") && ($input ne "y") && ($input ne
"n"));
# If the user enters no|n, exit.
if (($input eq "no") || ($input eq "n")) {
    exit 1;
}
print "-----\n";
# Remove the NRPE daemon on each host listed
foreach(@ipAddresses) {
    # Set destHost as removal host
    $destHost = $_;
    print "\nRemoval started on $destHost...\n";
    # Stop the NRPE daemon whether or not it is running.
    print "Stopping the NRPE daemon...";
    # MODIFY - SSH and stop the daemon
    # Remove group/user on host
    print "Removing group nagcmd and member (user) nagios...";
    # MODIFY - SSH and remove the user/group
    print "DONE\n";
    # Removing NRPE
    print "Removing NRPE...";
    #MODIFY - SSH and remove NRPE files
    print "DONE\n";
    print "...NRPE removed from ${destHost}.\n";
}

```

```

}
print "\n-----\n";
print "Removal finished successfully.\n";
exit 0;

```

F.4 NRPE Manager

This is a generalized template of a script created to start, stop, and restart NRPE on destination hosts.

Change anything listed in CAPITAL letters (uppercase), any paths to files, anything listed with the text of "MODIFY: ", and anything else where necessary.

Code

Example F-4 is the code for the NRPE manager.

Example: F-4 NRPE manager

```

#!/usr/bin/perl
#####
# File Name: nrpeManager.pl
# Author: Blue Gene Speed Team
# Date: Summer 2007
#####
#Description - This script is used to start, stop, and restart on any hosts given
#as input.  #
#
#Usage - All hosts that you wish to start, stop, or restart NRPE on, must be
#entered in a file with the path of nrpemasterfolder/hostips. Each host must be
#listed on a separate line.#
#####
my $ipFile = "PATH_TO_HOSTIPS_FILE";
my $ipAddresses;
my $destHost;
my $action;
my $actioning;
my $actioned;
# Read in IP addresses to manage
open(DAT, $ipFile) || die("Could not open \"${ipFile}\" file. Please list the IPs
you wish to manage in a file \"${ipFile}\"");
@fileData=<DAT>;
chomp(@fileData);
close(DAT);
# Give the user the option screen
print "\nNRPE Host Manager - Options:\n";
print "1 - Start Hosts\n";
print "2 - Stop Hosts\n";
print "3 - Restart Hosts\n";
# Ask the user what they wish to do
do {
    print "\nPlease select an option from above to perform on all hosts
[1-3]:";
    $action = <STDIN>;

```

```

        chomp($action);
    } while (($action ne "1") && ($action ne "2") && ($action ne "3"));
    # Set up other UI variables
    $actioning = $action;
    $actioned = $action;
    print "-----\n";
    print "Reading file \"${ipFile}\" for the list of destination hosts...\n";
    # Strip out any blank lines and comments (#)
    @ipAddresses = grep { $_ ne "" } @fileData;
    @ipAddresses = grep { m/^[^#]/ } @ipAddresses;
    # Print each server inputed
    foreach(@ipAddresses) {
        print "$_\n";
    }
    # Replace selection with text
    $action =~ s/1/Start/gi;
    $actioning =~ s/1/Starting/gi;
    $actioned =~ s/1/started/gi;
    $action =~ s/2/Stop/gi;
    $actioning =~ s/2/Stopping/gi;
    $actioned =~ s/2/stopped/gi;
    $action =~ s/3/Restart/gi;
    $actioning =~ s/3/Restarting/gi;
    $actioned =~ s/3/restarted/gi;
    # Make the user confirm the process for the servers. Continue looping until
    yes/no/y/n is entered.
    do {
        print "\n${action} NRPE on the above servers? <y|n>:";
        $input = <STDIN>;
        chomp($input);
    } while (($input ne "yes") && ($input ne "no") && ($input ne "y") && ($input ne
    "n"));
    # If the user enters no|n, exit.
    if (($input eq "no") || ($input eq "n")) {
        exit 1;
    }
    print "-----\n\n\n";
    # Setup the NRPE daemon on each host listed
    foreach(@ipAddresses) {
        # Set destHost as installation host
        $destHost = $_;
        # Converting operation to lowercase
        $action = lc($action);
        print "${actioning} $destHost...";
        # MODIFY: SSH to destination host and start nrpe by passing it the
        ${action} variable.
        print "DONE\n\n";
    }
    print "\n-----\n";
    print "All hosts have been ${actioned}.\n";
    exit 0;

```

Archived

Additional material

This paper refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this paper is available in softcopy on the Internet from the IBM Redbooks publications Web server. Point your Web browser at:

<ftp://www.redbooks.ibm.com/redbooks/REDP4356>

Alternatively, you can go to the IBM Redbooks publications Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redpaper form number, REDP4356.

Using the Web material

The additional Web material that accompanies this paper includes the following files:

<i>File name</i>	<i>Description</i>
bgscrip.tgz	Zipped code samples

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: Under 1 MB

How to use the Web material

Create a subdirectory (folder) on your workstation and unzip the contents of the Web material zip file into this folder using **taz**.



Deployment Guide for Advanced Monitoring of a Blue Gene



See the IBM Blue Gene solution overview

Learn to install and configure a Blue Gene environment

Take advantage of code scripts for plug-ins

The larger a data center grows, the more monitoring it requires. But limited manpower, mixed hardware, multiple software configurations, and complex network topologies all combine to increase the complexity of monitoring. In addition, high-dollar monitoring solutions require cost justification, additional training, higher software budgets, and expensive maintenance contracts, and they can add their own idiosyncrasies to a complex environment. Adding more people to provide 24/7 operational coverage is not cost-effective, except in the largest computing environments.

How can we make the most effective use of limited technical staff and alert the proper resource to occasional but critical system errors or potential problems in the making and enhance troubleshooting? In addition, every data center is faced with updated architectural requirements and unanticipated configuration changes. Constant requests for new hardware and software requirements must be weighed against limited budgets and the need for uptime.

Nagios offers a flexible solution: a widely supported and extensible monitoring application. And it is free. Teaming Nagios up with the renowned IBM® Blue Gene® system creates an attractive combination.

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