IBM

IBM High Performance Computing Cluster Health Check

- Learn how to verify cluster functionality
- See sample scenarios
- Understand best practices

Dino Quintero
Ross Aiken
Shivendra Ashish
Manmohan Brahma
Murali Dhandapani
Rico Franke
Jie Gong
Markus Hilger
Herbert Mehlhose
Justin I. Morosi
Thorsten Nitsch
Fernando Pizzano

Redbooks

ibm.com/redbooks

International Technical Support Organization

IBM High Performance Computing Cluster Health Check

February 2014

SG24-8168-00

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

First Edition (February 2014)

This edition applies to Red Hat Enterprise Linux 6.3, IBM Parallel Environment (PE) (Versions 1.3.0.5, 1.3.0.6, and 1.3.0.7), GPFS 3.5.0-9, xCAT 2.8.1 and 2.8.2, Mellanox OFED 2.0-2.0.5 and 1.5.3-4.0.22.3, Mellanox UFM 4.0.0 build 19, iperf 2.0.5, IOR version 2.10.3 and 3.0.1, mdtest version 1.9.1, stream version 5.10.

Contents

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 grant 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites 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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

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

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

AIX®	IBM®	PureFlex™
AIX 5L™	iDataPlex®	Rational®
BladeCenter®	Intelligent Cluster™	Redbooks®
developerWorks®	Jazz™	Redbooks (logo) ®
eServer™	POWER®	System i®
Global Technology Services®	Power Systems™	System p®
GPFS™	POWER7®	System x®
HACMP™	PowerHA®	SystemMirror®

The following terms are trademarks of other companies:

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

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

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

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

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

Preface

This IBM® Redbooks® publication provides information about aspects of performing infrastructure health checks, such as checking the configuration and verifying the functionality of the common subsystems (nodes or servers, switch fabric, parallel file system, job management, problem areas, and so on).

This IBM Redbooks publication documents how to monitor the overall health check of the cluster infrastructure, to deliver technical computing clients cost-effective, highly scalable, and robust solutions.

This IBM Redbooks publication is targeted toward technical professionals (consultants, technical support staff, IT Architects, and IT Specialists) responsible for delivering cost-effective Technical Computing and IBM High Performance Computing (HPC) solutions to optimize business results, product development, and scientific discoveries. This book provides a broad understanding of a new architecture.

Authors

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

Dino Quintero is a complex solutions project leader and IBM Senior Certified IT Specialist with the ITSO in Poughkeepsie, NY. His areas of expertise include enterprise continuous availability, enterprise systems management, system virtualization, technical computing, and clustering solutions. He is an Open Group Distinguished IT Specialist. Dino holds a Master of Computing Information Systems degree and a Bachelor of Science degree in computer science from Marist College.

Ross Aiken is an Emerging Technologies Solution Architect for IBM Systems and Technology Group (STG) Worldwide Software Defined Systems. He has extensive experience in High Performance Computing (HPC) optimization and large file system scalability performance, for both x86-64 and IBM Power-based cluster architectures. His primary focus is on working with clients to develop solution requirements around emerging technologies. However, he often manages first-of-a-kind field integration plans, identifying and mitigating technical risks throughout clients' first testing, and then supporting them on deployments. He holds a B.S. in statistics from Purdue University, an M.P.A in public administration from Purdue, and a DPhil in political science from Nuffield College at the University of Oxford.

Shivendra Ashish is a software developer in IBM PowerHA® Development, and holds a master's degree in computer science and applications. He also worked across various Power virtualization and clustering technologies. He has over six years of experience working on these technologies. He also has extensive experience in configuring and managing IBM SAN storage subsystems, and has worked on various client use cases and environment by using his expertise.

Manmohan Brahma has been with IBM for the past 3 years. He works as an Architect for High Performance Computing with the IBM System and Technology Group in Bangalore, India. His team focuses on the design of HPC solutions for customers across India and South Asia. He has designed major supercomputing systems such as the fastest supercomputing system in India, to be commissioned at the Ministry of Earth Sciences and VIRGO, commissioned at the Indian Institute of Technology, Madras. Apart from IBM, he has worked as a research scientist of HPC applications in various Government of India research and development (R&D) labs, which include Council of Scientific and Industrial Research (CSIR) Fourth Paradigm Institute and Defence Research & Development Organization (DRDO). He holds a bachelor's degree in computer science & engineering from Roland Institute of Technology, India. He is very dedicated and enthusiast towards his major areas of interest HPC, cloud, and distributed computing.

Murali Dhandapani is a Certified IT Specialist in Systems Management with IBM India. He is working for the IBM India Software Lab Operations team, where he is a technical lead for IBM Rational® Jazz™ products infrastructure, high availability, and disaster recovery deployment. He has 10 years of experience, and his areas of expertise include Linux, IBM AIX®, IBM POWER® Virtualization, PowerHA SystemMirror®, System Management, and Rational tools. Murali has a Master of Computer Science degree. He is an IBM developerWorks® Contributing Author, IBM Certified Specialist in IBM System p® administration, and an IBM eServer™ Certified Systems Expert - pSeries High Availability Cluster Multi-Processing (IBM HACMP™).

Rico Franke is an IT Specialist for x86, IBM PureFlex™, and HPC Systems in Germany. He has more than 10 years of experience in supporting IBM System x® and IBM System i® solutions. Currently, he leads the operational support team at the Leibniz Supercomputing Centre, servicing the warm-water cooled SuperMUC. He holds a degree of engineering in information technology.

Jie Gong is a Software Engineer in Cluster System Test for HPC in the IBM China Systems and Technology Laboratory (CSTL). Since joining the team two years ago, he has worked with the IBM HPC Clustering with IBM Power 775, and IBM HPC Clustering with the InfiniBand switch and IBM Flex Systems. He has eight years of experience with IBM Power Systems™. He received Red Hat Certified Engineer (RHCE) certification in 2003.

Markus Hilger has been a corporate student of business information technology in Berlin, Germany since 2011. He has experience in supporting System x and programming. He has worked in the operational support team at the Leibniz Supercomputing Centre (LRZ) for 3 months, and developed a cluster health check tool for it. He helped develop the IBM HPC cluster health check tool.

Herbert Mehlhose is an IT Specialist working for IBM Germany. He has been with IBM for 29 years, working in many different areas, including networking and storage hardware and software. Currently, he is concentrating on HPC clusters working for IBM Global Technology Services® (GTS), and has been a member of teams responsible for large cluster installations since 2012.

Justin I. Morosi is a Consulting IT Specialist working for IBM STG as a Worldwide Solution Architect. He has worked for IBM for over 15 years, and has more than 22 years of consulting and solution design experience. He holds both a bachelor's and master's degree from California Polytechnic State University - San Luis Obispo. He also holds numerous industry-recognized certifications from Cisco, Microsoft, VMware, Red Hat, and IBM. His areas of expertise include HPC, storage, systems management, and virtualization.

Thorsten Nitsch is an IT Specialist working for IBM Germany. He joined an IBM subsidiary in 2001, and has worked with Linux-based and UNIX-based data center infrastructures since then. During the past six years, he has worked mainly with IBM SAP appliances, such as SAP HANA and SAP BWA, from planning to implementation in client projects. In addition, he worked with a large BlueGene/P HPC cluster at a large German research center, and was part of the IBM Systems solution for SAP HANA development team. Thorsten holds Linux Professional Institute (LPIC-3), RHCE, and Novell Certified Linux Professional (NCLP) certifications, and is a SAP Certified Technology Associate - SAP HANA 1.0.

Fernando Pizzano is a hardware and software bring-up (the initial deployment of a cluster) Team Lead in the IBM Advanced Clustering Technology Development Lab, Poughkeepsie, New York. He has over 10 years of IT experience, the last five years in HPC development. His areas of expertise include AIX, pSeries High Performance Switch, and IBM System p hardware. He holds an IBM certification in pSeries AIX 5L™ System Support.

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

Ella Buslovich
International Technical Support Organization, Poughkeepsie Center

Mark Atkins, John Lewars, John Dorfner
IBM US

Ravindra Sure
IBM India

Torsten Bloth
IBM Germany

Now you can become a published author, too!

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

- Use the online **Contact us** review Redbooks form:

  ibm.com/redbooks

- Send your comments in an email:

  redbooks@us.ibm.com

- Mail your comments:

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

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

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

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

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

- Stay current on recent Redbooks publications with RSS Feeds:

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

1

Introduction

This IBM Redbooks publication mainly provides information about IBM High Performance Computing (HPC) clusters. Therefore, in the rest part of this book, unless otherwise mentioned, the term *cluster* represents an IBM HPC cluster. A computer cluster is a group of connected computers that work together. In many respects, they act as a single system.

We describe the concepts, processes, and methodologies used to achieve and maintain a “healthy” state for an IBM HPC system, both in pre-production stage and production stage.In the context of this book, *healthy* means *performing as expected* or *well-configured and working correctly.*

The following topics are described in this chapter:

- Overview of the IBM HPC solution
- Why we need a methodical approach for cluster consistency checking
- Tools and interpreting their results for HW and SW states
- Tools and interpreting their results for identifying performance inconsistencies
- Template of diagnostics steps that can be used (checklists)

1.1 Overview of the IBM HPC solution

HPC, also known as *supercomputing* or *Technical Computing*, is implemented using computers that have the fastest contemporary processing capacity. Usually, all of the computers in an HPC cluster are connected tightly with a high-throughput network.

For many years, IBM provided HPC solutions in different scales that provided extreme performance for demanding workloads, including weather forecasting, petroleum reservoir modeling, computational chemistry, and so on. These systems were used in different types of industries.

1.2 Why we need a methodical approach for cluster consistency checking

The whole HPC cluster is a complex system that involves a variety of hardware and software. Different types of computer architectures, network intercommunication techniques, storage equipment, operating systems, and administrative tools are used in an HPC environment. Because of this complexity, you need to carefully plan the installation and *bring-up* (the initial deployment of a cluster) to ensure that the deployment and maintenance work can be performed smoothly, and that mistakes can be prevented or recovered from quickly.

There are additional considerations:

- The HPC cluster systems are expensive.
- They use a large amount of electricity.
- Most of the hardware components require appropriate cooling systems.

  Some of the HPC cluster systems use water-cooling technologies that enable a significant portion of the energy spent on HPC to be recovered in the form of chilled water, which can then be used to cool other parts of the computing center.

To implement a fully working HPC cluster system, all of these different components require correct deployment, and they need to work together. Usually, it takes a group of people to work for dozens of weeks to complete the full deployment of a complex cluster system.

People who are planning and organizing the cluster system should use a cluster *health check* methodology to improve the efficiency of deployment and maintenance, and to prevent problems as well. For the system administrators who work on cluster maintenance daily, we provide advice and guidelines for cluster health checking and verification. We also provide suggestions for which tools to use.

1.3 Tools and interpreting their results for HW and SW states

You use tools to verify hardware and software states and configuration. Normally, the health check involves a series of examination and verification steps. You need to keep uniformity among different individual computers within a cluster system. Usually, thousands of individual computers are involved in one HPC cluster system. To check them manually, one by one, is not a good approach, because manual operations are needlessly time-consuming and error prone. Therefore, some kind of automated checking and verification facilities are necessary.

However, for a production implementation, a single, fully automated checking and verification facility for an HPC cluster system does not exist. This is because it is too difficult to develop such a comprehensive facility. There are too many different computer architectures, different types of operating systems, different types of input/output (I/O) adapters, and different types of storage systems.

To implement such a fully automated checking and verification facility, the verification tool must be tested and verified in an environment that includes various combinations of all of these types of different components. This is impossible, because there are too many possible combinations. Based on these reasons, the administrators of an HPC cluster system should implement their own checking and verification tools, to use and adapt existing tools to their unique cluster environment.

Most of the contents of this book describe some fundamentals about maintaining and preserving a healthy state for an HPC cluster system. We explain the following concepts:

- What to do
- Why to do these things
- Examples illustrating certain points

Therefore, based on these principles, system administrators can develop their own checking and verification tools.

In the latter part of this chapter, we introduce a few simple tools used in the HPC cluster environments. Some of these tools are part of the HPC system software, and some of them are actually HPC cluster management tools. We introduce them here because they appear in most of the later chapters and examples.

1.3.1 General Parallel File System

The IBM General Parallel File System (IBM GPFS™) is a high-performance, shared-disk file system that provides data access from all nodes in a homogeneous or heterogeneous cluster of IBM UNIX servers running either the AIX or the Linux operating system.

1.3.2 Extreme Cloud Administration Toolkit

IBM Extreme Cloud Administration Toolkit (xCAT) is open-source cluster computing management software used for deployment and administration:

- Provisions operating systems on different architecture:
  - Linux
  - AIX
  - IBM System x
  - IBM Power Systems
- Creates and manages clusters
- Installs and manages many cluster machines in parallel
- Remotely manages the system:
  - Remote console
  - Distributed shell

For additional information see the xCAT website:

http://xcat.sourceforge.net/

1.3.3 The OpenFabrics Enterprise Distribution

The OpenFabrics Enterprise Distribution (OFED) is open-source software for RDMA and operating system kernel bypass application. OFED includes kernel-level drivers, channel-oriented RDMA and send/receive operations, operating system kernel bypasses, kernel-level application programming interfaces (APIs) and user-level APIs.

OFED is available for many Linux distributions, including Red Hat Enterprise Linux (RHEL), SUSE Linux Enterprise Server (SLES), and so on.

OFED includes many user-space tools that can be used to verify and diagnose the state of InfiniBand interfaces and topology.

For additional information, see the OpenFabrics website:

http://www.openfabrics.org/

For additional information about the Mellanox-enhanced version OFED, see the Mellanox website:

http://www.mellanox.com/page/products_dyn?product_family=26

1.3.4 Red Hat Package Manager

The Red Hat Package Manager (RPM) is a powerful package management system capable of installing, removing, querying, verifying, and updating software packages. Each software package consists of an archive of files, along with information about the package.

RPM is widely used in different Linux distributions, including RHEL, SLES, and others. RPM is also a built-in component of the IBM AIX operating system.

The RPM archive format is an official part of the Linux Standard Base (LSB):

http://linuxfoundation.org/collaborate/workbroups/lsb

RPM is released as no-charge software under the GNU Lesser General Public License (LGPL) distribution license:

http://rpm5.org/

1.4 Tools and interpreting their results for identifying performance inconsistencies

As we mentioned in the previous section, there is no fully automated tool for HPC cluster system checking and verification. In addition, no such automated tool exists for interpreting system checking results and diagnostics.

In a complex, gigantic cluster environment, sometimes the point of failure can be detected and fixed easily. However, most of the time it is difficult to detect and fix.

1.5 Template of diagnostics steps that can be used (checklists)

For quick reference, this section includes a few example checklists for basic verification, checking, and diagnostics. Before you diagnose your problem, we suggest that you go through these checklists.

Management nodes

Use the following checklist for management nodes:

- ☐ There are at least two management nodes for redundancy (optional but suggested).
- ☐ There is enough available disk space for service nodes, compute nodes, and utility nodes.
- ☐ The clock and time zone settings are correct, and there is a utility to maintain them. A management node can act as a Network Time Protocol (NTP) server.
- ☐ The hardware connections are functional in an xCAT environment. Use the **`lshwconn -l`** command to verify this.
- ☐ There are redundant Ethernet connections.

Service nodes (if any)

Use the following checklist for service nodes:

- ☐ There are at least two service nodes for redundancy.
- ☐ The service nodes can be accessed from the management node using Ethernet.
- ☐ There is enough available disk space for compute nodes.
- ☐ The clock and time zone settings are correct, and an NTP client is synchronized to an NTP server.

Ethernet switch

Use the following checklist for an Ethernet switch:

- ☐ All of the Ethernet switch LEDs of ports with a cable on them are flashing.
- ☐ All switches have remote execution configured from xCAT (you can run **`xdsh`** to them).
- ☐ All switches have the correct clock and time zone setting, and an NTP client is synchronized to an NTP server.

InfiniBand switch (if any)

Use the following checklist for an InfiniBand switch:

- ☐ All of the LEDs of ports with a cable on them are flashing.
- ☐ None of the service LEDs indicate a problem (shown in the switch hardware documentation).
- ☐ All switches are pingable and Secure Shell (SSH)-able from the management server.
- ☐ The naming convention for each switch is based on its physical location.
- ☐ All switches of the same type are running at the same firmware and software levels.

InfiniBand Unified Fabric Manager (if any)

Use the following checklist for InfiniBand Unified Fabric Manager (UFM):

- ☐ No errors are displayed on the UFM dashboard.
- ☐ The UFM health check function returns no errors.

- ☐ The UFM dashboard traffic monitor shows average congestion is less than maximum congestion.

Compute nodes

Use the following checklist for compute nodes:

- ☐ The naming convention for each node is based on their physical location.
- ☐ All of the compute nodes have the same hardware configuration.
- ☐ All of the compute nodes run the same operating system level.
- ☐ The software packages on different compute nodes are the same.
- ☐ The clock and time zone settings are correct, and an NTP client is synchronized to an NTP server.
- ☐ The version levels of all of the software packages on different compute nodes are the same.
- ☐ Each of the compute nodes can ping every other compute node successfully through Ethernet.
- ☐ Each of the compute nodes can ping every other compute node successfully through InfiniBand.

Utility nodes (if any)

Us the following checklist for utility nodes:

- ☐ There are at least two utility nodes for redundancy.
- ☐ The clock and time zone settings are correct, and an NTP client is synchronized to an NTP server.

Login node (if any)

There are at least two login nodes for redundancy.

Hardware Management Console (if any)

Use the following checklist for Hardware Management Consoles (HMCs):

- ☐ There are at least two HMCs for redundancy.
- ☐ All physical machines can be accessed from the HMC.
- ☐ Each physical machine has at least one logical partition (LPAR) on it.
- ☐ There is no alert from the HMC.

2

Key concepts and interdependencies

This chapter provides key concepts and interdependencies. The following topics are covered in this section:

- Introduction to High Performance Computing
- Rationale for clusters
- Definition of an HPC Cluster
- Definition of a “healthy cluster”
- HPC preferred practices

2.1 Introduction to High Performance Computing

In its most simple form, a modern multi-processor supercomputer is just a high-speed network of computers working together in parallel to solve a problem. The reality is quite a bit more complex, full of new technologies and perhaps even using innovations in building facilities not often associated with traditional computer science.

For example, in SuperMUC, hot water cools 150,000 processor cores to provide a peak performance of up to three petaflops, which is equivalent to the work of more than 110,000 personal computers working in unison.

Building clustered systems at this scale is indeed a daunting task, especially if you don't first ensure that they can be properly configured and tuned to provide trouble-free operation and optimal performance. As High Performance Computing (HPC) cluster environments grow to many thousands of processors, hundreds of subtle configuration and design decisions made early in the process become critical to ensuring overall system reliability.

In addition, although the difficulty of bringing up just such an HPC cluster seems intimidating, many cluster problems are in fact easily resolved through careful verification steps.

IBM has built enough of the world's largest HPC clusters to know that if teams make poor configuration choices or decisions that are not in compliance with a preferred practices design during deployments, they can become more difficult to find and correct as the system scales in terms of nodes, applications, and users. The following sections describe a toolkit that can aid system administrators in the initial starting stages of installing their own cluster.

2.2 Rationale for clusters

So, if you're not out to build the world's largest HPC clusters, why did you pick up this IBM Redbooks publication? Probably because, even if your cluster system environment is small today, this might be the ideal time to lay a solid foundation that helps ensure that your computing environment can scale up with the growing needs of your systems.

The reader will see the term *cluster health* repeated often in the subsequent chapters. Cluster health describes a correctly working cluster:

- Deployed in accordance with leading practices
- Had settings configured and confirmed uniformly
- Demonstrated that major components are individually working as expected
  Identified and isolated any components that do not meet expectations

At this point the cluster system is presumed capable of uniformly and repeatedly yielding an expected outcome.

2.3 Definition of an HPC Cluster

To reduce these complex systems into more common terms, we all too often use jargon to refer to them. *HPC cluster* is spoken of throughout the halls of almost every major research university today. HPC people often refer to the individual computers within a cluster as *nodes*. This type of technical jargon actually helps understanding. It relates something by using a metaphor. The jargon becomes a figure of speech in which an implied comparison is made between two unlike things that actually have something in common.

Using this language trick, thinking about all of the common components that are in HPC clusters is an easy way to help understand what they are. You might be surprised to learn that SuperMUC systems have all of the elements you would find on your own personal notebook (processors, memory, disk, and operating system) just many more of them. Each individual node in a commonly configured HPC cluster has one (four processors), and today's processor technologies typically have two (eight cores).

There is at least one very distinct component of an HPC cluster that you might not commonly find on your personal computer. The history of HPC clusters closely parallels the development of high speed - low latency networks. Before we jump into the complexity of these cluster interconnect fabrics, let us again make a personification of the typical HPC cluster, in which this inanimate system is given human qualities or abilities.

Similar to people, the nodes need to be able to talk to one another to work in a coordinated and meaningfully way, in unison. That is a simplistic, but hopefully meaningful, description of interconnect fabrics. For many readers, that definition does not suffice. For them, there follows a more precise network technical jargon that they expect.

InfiniBand is a type of network communications link for data flow between processors and I/O devices that offers throughput of up to multiple gigabytes per second (GBps), and support for up to 64,000 addressable devices. Because it is also scalable and supports quality of service (QoS) and failover, InfiniBand is currently now favored as a cluster interconnect in high-performance clustering environments.

As the authors of this IBM Redbooks publication can attest, when people take on the task of interconnecting all of the nodes in an HPC cluster, these machines seem to take on human-like qualities, such as stubbornness, that at times can slow progress. There is also a risk of losing objectivity when technical people choose to assume that these machines possess any degree of human traits. Nevertheless, one more anthropomorphic explanation can help increase your understanding of HPC clusters.

In the early phases of bringing up a large cluster it is often chaotic before each node is checked and the cluster becomes *healthy*. It is a common and often repeated task for administrators to find nodes that are determined to be *unhealthy* and mark them as down or offline so as to enable the overall cluster deployment to continue.

When a node is bad, further problem determination is then done, but the general idea here is to prevent any jobs from being scheduled or run on these sick nodes. This helps increase the overall reliability and throughput of a cluster by reducing preventable job failures due to misconfiguration, known hardware failures, and so on.

2.4 Definition of a "healthy cluster"

The following characteristics summarize the concept of a healthy computing environment:

- Has been deployed in accordance with preferred practices
- Had its application and hardware settings uniformly configured and independently verified
- Can repeatedly process workloads within predetermined expectations

More succinctly stated, *a healthy compute environment is one that is performing as expected*. This does not necessarily mean that every component is operational, or that all of the components are capable of processing. It merely means that the components required to process workloads are functioning without error and working within acceptable criteria. Therefore, individual unhealthy components can exist within a healthy compute environment so long as they are isolated and not impacting overall health.

The intent of including unhealthy (or inconsistent) components in the overall health description of a compute environment's state is to acknowledge the fact that most computing environments are in a constant state of fluctuation. Components can be down for maintenance reasons, quarantined for problem determination, or merely shut off to save power.

The key difference between both a component's state and the overall compute environment's state is expectation. A node that has been shut down sets an expectation of not being operational. However, a node that has intermittent network problems might be operational, but still not meeting expectations.

A compute environment could be considered healthy or unhealthy in either of the previous two scenarios. If the expectation is to have 100 nodes available for processing workloads but one was shut down to save power, the compute environment could be deemed unhealthy. This is despite the fact that all nodes, network and storage work perfectly.

Similarly, the compute environment could be characterized as healthy if the expectation is merely to have it processing workloads without errors. Justifiable arguments can be made for either scenario, but ultimately the determination of health is derived from expectations.

2.5 HPC preferred practices

The phrase *preferred practices for HPC clusters* is used in the most general sense, to describe the practice of developing and following a standard way of accomplishing things that other practitioners can use. It is important to maintain realistic expectations when seeking a preferred practice, and to be wary of *internal validity problems*. Such problems include realizing that the research required to identify the absolute "best" practice for any broad use is almost never possible.

Our suggested practices are tailored for HPC use cases and based on observed results, but that does not necessarily mean they are always going to create good results. The suggested preferred practices used herein are part of the Redbooks publication program, providing a narrative of solutions that engineers have created while supporting our clients.

Because clusters can be used in a wide range of both known and custom scenarios to meet different business needs, this IBM Redbooks publication outlines and describes certain conditions that fit HPC use, and that do not warrant further review by IBM to determine whether a cluster is supportable.

An exception to a preferred practice does not necessarily mean that a cluster is not supportable by IBM. Although some deploying with exceptions can result in unsupportable scenarios, most other changes only need careful review and analysis before proceeding.

There are significant challenges concerning how to get to pertinent health data, which is in many layers of these clustered systems. Reporting and correlating critical system data, and transforming it into actionable information, is not trivial. Today's cluster health check tools mostly test components and a little point-to-point network performance. They are lacking in the following abilities:

- Comprehensively monitor and assess overall system health.
- Assess true aggregate performance.
- Reconcile conflicting use, user expectations, and demands.

In our initial release, this health check toolkit first attempts to establish a few reference baselines of performance, and verify prerequisite services at the component level. This enables productive testing and detection of anomalous conditions, and can even extend to periodic consistency checking at the component level when the tests are not invasive. This approach is particularly important when trying to isolate the cause of performance problems as initial deployment system testing uncovers them.

We do not intend to bash benchmarking and performance-based modeling as an alternative approach. Some researchers have had success with performance modeling results used as strict cluster bring-up and acceptance criteria. It does, however, extend the time it takes to bring a cluster up, and adds considerable expense.

Even those who are successful admit that there remain significant challenges in setting reasonable performance expectations for broadly diversified workloads on new HPC systems, to realistically set accurate expectations for performance at all stages of system integration.

3

The health lifecycle methodology

This chapter describes the requirement for a methodology to address the entire lifecycle of the compute environment components. This chapter guides you through the different phases of a cluster's life, and gives suggestions based on the experiences of the authors.

This chapter covers the following topics:

- Why a methodology is necessary
- The health lifecycle methodology
- Practical application of the health lifecycle methodology

3.1 Why a methodology is necessary

Due to the complexity, interdependence, and magnitude of potential differences within a compute environment, there needs to be a way of addressing and managing expectations. The simplest way would be to establish test and pass criteria for every expectation, but due to the multidimensional nature of compute environments this could become untenable quickly.

A more prudent approach is to employ a methodology that is dynamic enough to measure and evaluate expectations at any point in time. It then becomes the framework of the methodology that ensures that every component is always uniformly configured, operating correctly, and performing as expected.

An HPC cluster passes different stages during its lifetime, and the methodology leads through it by indicating tasks, procedures, and suggested practices. The method should also assist with defining operational procedures and responsibilities. These are mandatory for maintaining a healthy state during day-to-day operations, and are the foundation that ensures that recurring tasks are handled with constant care.

3.2 The health lifecycle methodology

The health lifecycle methodology is a holistic approach to managing an entire compute environment's consistency and stability. It consists of a collection of processes and procedures that evaluate the functionality and homogeneity of individual components.

The implicit assumption is that if the individual components are performing as expected, then their aggregate sum should yield similar results. This extrapolation is useful in terms of establishing stability and predictability for your environment, and also in terms of relative performance and troubleshooting.

Taking into account that compute environments are always in a state of flux, there is a need to quickly and easily identify unhealthy components within an otherwise healthy compute environment. The health lifecycle methodology does this by using a suite of tools to evaluate key expectations and identify outliers or inconsistent settings.

The expectations can include uniformity of settings, relative network performance, and memory bandwidth performance. Through measurement and evaluation, the health state of any component can be determined, or at the least compared.

Figure 3-1 on page 15 depicts a high-level overview of the health lifecycle methodology. Note the cyclical nature of verification, which continues until expected performance is achieved.

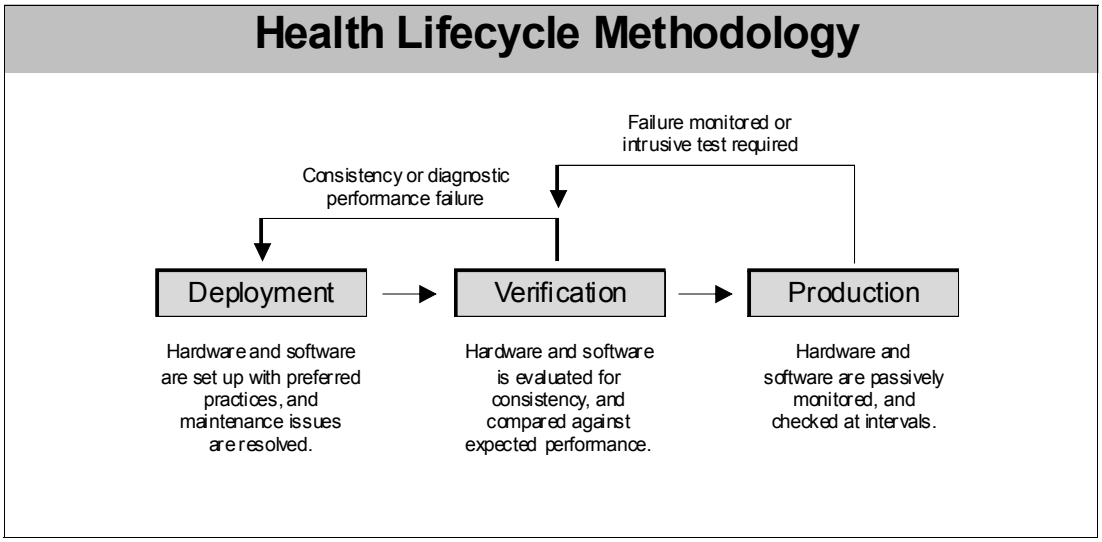

Figure 3-1 Diagram of the health lifecycle methodology

The benefit of using the health lifecycle methodology is that it provides a defined set of repeatable steps and tools that ensure consistency and stability. The focus of this book is on the methodology at it pertains to reducing initial environment deployment time, minimizing efforts required for health state checking, and the tools that assist in decreasing performance variation. The actual steps, and the corollary tools, are described later in this book.

3.3 Practical application of the health lifecycle methodology

From buildup to final decommissioning, a cluster goes through different phases during its lifetime in iterative cycles. Updates, changes, maintenance, or outages require methods to run through the stages consistently. Each phase has its own processes, characteristics, and needs. The goal of the methodology is to reduce nonproductive time to a minimum.

The lifecycle starts with the *deployment* stage, where all the different components are assembled, connected, powered, installed, and configured. While the cluster grows, it has to be carefully and continuously examined. Basic functions are verified, and uniformity between similar devices has to be ensured.

Discrepancies and abnormalities must be analyzed and eliminated. Otherwise, they increase with the size of the clusters, and ultimately result in negative impacts to the ability to meet expectations. IBM publishes preferred practices and suggestions to prevent you from running into known problems and traps.

More and more components are collated and tested until the whole cluster is ready for *verification*. During that phase, you have to prove that a cluster is able to perform estimated workloads and meet expectations. That is the time to confirm functionality and performance through extensive testing and evaluation.

Verification starts with the smallest entity in the cluster, and then scales to the whole cluster size. For example, it is a good practice to prove the performance of each single compute node before running a test at large. Reducing the size and complexity to a minimum is one of the basic rules about health. If a test fails, the source of the failure has to return to the deployment stage until it is able to meet the requirements.

Finally, after all of the inspections are successful, a cluster reaches the *production* phase and has to fulfill its function, which is to run user jobs and return results. To keep the cluster at that state, it is mandatory to monitor all key characteristics continuously. Failures or discrepancies from standards must be quickly identified, the cause investigated, and the source isolated.

If it is not possible to separate the cause of a problem, then the cluster on the whole must be verified again. In addition, if a system-wide change is required to solve a problem, it starts again at the beginning in the deployment phase. In other words, insufficient results in one of the stages causes a start over at the previous phase.

However, the lifecycle does not describe only the phases of the cluster. It also applies to each single component. There is no difference between a large cluster or a single device when it comes to changes. So even if a single cable has to be replaced, the old one must be isolated from the remaining cluster to do no harm. Afterward, a new cable has to be deployed according to preferred practices and carefully verified before it goes to production again.

It is an HPC rule that the slowest device represents overall performance. Therefore, an extensive verification and continuous monitoring is crucial to obtaining and maintaining a healthy cluster. After a maintenance action, there is no excuse for overlooking one poorly performing DIMM that throttles down all parallel jobs.

It is important to have tests in place to ensure that all critical components work according to their expectations. Most of the testing should be done before combining them with other similar components. In the following chapter, we present a few tests that should be useful in your cluster as well.

3.3.1 Deployment phase

The initial deployment of a cluster is often called *bring-up*. In that fast-paced time, all of the components are delivered and assembled onsite. A careful planning of schedules and activities is required to optimize the installation time. Many different teams have to work together, so you also want quick and open communication between them.

Deployment of a cluster

Typically, the bring-up starts with installation of the management server, together with core network and storage. These are vital parts, and it is necessary to power on, configure, and install all of the other components.

Because all of the other parts of the cluster depend on this core infrastructure, any outage discovered at a later time has a direct effect on the installation activities of the entire cluster. In terms of the lifecycle, the core infrastructure should already be at a production stage before deploying the compute environment.

Important: Build up the management infrastructure ahead of the remaining cluster. Plan enough time to customize, verify, and stabilize. Did you test backup, failover, and recovery? If any disruptive activity does not happen directly at the beginning, it will most likely not take place. After the cluster implementation starts, it is almost impossible to introduce changes on the management server. Also, after the cluster is in production, nobody is happy about a maintenance window (downtime) for a failover test.

A cluster is usually built up in small entities, such as rack by rack. After a compute rack is connected to the network and power, the hardware can be configured and examined. Afterward with the start of the operating system image, that portion is ready to get validated.

In that stage, it is sometimes impossible to determine if a component is in the deployment or verification stage, because it swings forward and backward in between as problems are discovered and resolved. After a configuration is applied, it has to be verified before starting the next installation step. In iterative cycles, each component, in addition to the whole cluster, converges to a stable condition and can be validated in its entirety.

The following practices can help with the cluster bring-up approach:

- Read preferred practice guides.
- Bring the entire team together, and plan responsibilities and dependencies on one another.
- Agree upon standard communication methods, and methods to hand-off responsibilities.
- Start working on delimited groups of cluster devices, such as a frame or a rack.
- Deploy changes in defined steps and orders, for example: update firmware, change hardware settings, start the operating system, joining InfiniBand.
- Verify each deployment step afterward, and make sure that all functional criteria are met at the current stage.
- Deployment steps should be large enough to gain progress, but not so large that sudden errors cannot be traced back to the last change.
- Each node must undergo extensive testing before it is allowed to join common resources, such as the file system, Ethernet, and InfiniBand networks.
- Define a schema to flag and isolate components if a test fails, so that these do not delay the deployment of group members.
- Unverified components must be kept as isolated as possible, to avoid side effects on other systems that can spread through network connections.
- Deployment activities must use the system management infrastructure, and all actions should be done as parallel as possible. This minimizes the time required, and the number of errors introduced by repetitive tasks.

At a later time, the cluster returns to the deployment stage during maintenance. The planning of maintenance requires a trade-off between the number of changes to deploy and the time required, versus the number of maintenance windows required. To minimize unproductive cluster time, maintenance activities are often accumulated in large chunks. This introduces an inherent risk of unforeseeable results, because these changes might not have been tested at large scale before.

A cluster is a complex entity, and all possible impacts of a change can hardly be predicted. If a later malfunction cannot be traced back to a certain change activity, the whole maintenance has to roll back. This raises the further point that there should always be a plan for a rollback in place.

The following practices can help with the cluster maintenance approach:

- Review preferred practice guides for updates.
- Verify if software or firmware changes are approved for your cluster (see the *best recipe*).
- Do not overload maintenance windows with changes.
- Provide time for removing changes should something go wrong.
- Keep records of all planned changes, the progress, and the results.
- Validate that all the changes come into effect.
- Validate that unchanged cluster components behave as they did before.
- Retain and document the actual cluster state for rollback purposes, but also for comparing with the new cluster state.

The purpose of a High Performance Computing (HPC) cluster is to solve scalable problems in a shorter time through parallelism. It is worth mentioning that improper setup or small inconsistencies are affected by cluster scalability just as correct configuration and consistency does. The improper results are often magnified by scale. To avoid these types of obstacles, IBM publishes preferred practices and best recipes for HPC clusters, and updates them frequently. Therefore, a cluster should be deployed according to these documents.

Note: Preferred practices are published at IBM developerWorks in the HPC Central Wiki:

https://www.ibm.com/developerworks/community/wikis/home/wiki/Welcome%20to%20High%20Performance%20Computing%20(HPC)%20Central

Best recipe refers to a suggested set of cluster-wide software and firmware levels that have been verified to operate together. These supported combinations can be found on HPC Central Cluster Service Packs, which might refer to other best recipe sites as well:

https://www.ibm.com/developerworks/community/wikis/home/wiki/Welcome%20to%20High%20Performance%20Computing%20(HPC)%20Central/page/IBM%20High%20Performance%20Computing%20Clusters%20Service%20Packs

Best recipe files are available for download at IBM Fix Central (for example, by searching for *Intelligent Cluster*):

http://www-933.ibm.com/support/fixcentral

Deployment of a component

An HPC cluster accumulates a huge number of components, so it is to be expected that a few of them can fail during day-to-day operation. For overall cluster health, the negative effect is not critical most of the time, *if* a failing device can quickly be identified and isolated. The fault can then be compensated by redundancy or fault tolerance. For instance, if a disk drive fails, it can be covered by the disk subsystem through RAID protection techniques. If a compute node or a single network link goes down, a job starts on another one.

But what does this mean for the failing component? From the lifecycle perspective, it first moves to the verification phase to determine the reason for the fault, and then moves to the deployment phase to get repaired, replaced, updated, or reconfigured. Because the remaining cluster components are still in operation, it requires much more care than the initial deployment. Any side effects to shared resources, such as file system, network, or InfiniBand, can cause other components to fail, or the overall cluster expectations to no longer be met.

Therefore, it is mandatory to have isolation procedures in place. Powering off troubled compute nodes or unplugging broken network cables are obvious actions. Switches or part of the management infrastructure must be isolated with care, because of their dependencies with other devices. Sometimes, it is necessary to delay the isolation of critical components to avoid even more negative impacts.

This is because, even if a critical device is redundant, a deactivation might account for rerouting activities or InfiniBand heavy sweeps that causes an unwanted interruption of network traffic and job communication. In that case, the skilled operator has to weigh which action affects the cluster less. For example, if a link is running at a slower speed, you might not unplug it if it is in the middle of the network, but you might isolate it if it is the only link into a particular node.

The following practices can help with the component maintenance approach:

- Isolate the component from the remaining cluster.
- Define a schema to distinguish between healthy and unhealthy compute nodes.

- Keep track of the failures for each component, to identify recurring errors.
- Perform diagnosis and repair actions in a safe condition or sandbox.
- Establish criteria, tests, and measurements to validate if a component can safely rejoin the cluster.
- Increase monitoring for the particular component after it has rejoined the cluster.

It is a suggested practice to establish operational procedures about the handling of suspicious or faulty components:

- How do you distinguish between a good or a bad compute node?
- How do you make sure that user jobs run only on validated nodes?
- How do you notify the responsible technician to take care of a failed device?
- How do you keep a log file of all maintenance activities and root causes?

Even if these operational procedures are not handled within this book, we suggest defining necessary workflows around repair activities.

3.3.2 Verification or pre-production readiness phase

Verification is the small but highly important phase between deployment and production. From a single device to the whole cluster, it determines if a user job can run reliably. Any faulty or misconfigured component that slips through will certainly cause trouble, which usually means job cancels or degraded performance. Careful examination must ensure that all of the devices of the same type are in a confirmed, consistent, and uniform condition. The goal of verification is to gain confidence in hardware and software before introducing it to the user.

For verification, we must understand the applicable parameters and final expectations. These are sometimes obvious, such as *all error indicators are supposed to be off*. A few settings are strongly suggested in the published preferred practices and the cluster service pack. For example, in System x-based clusters, you find favored integrated management module (IMM) and Unified Extensible Firmware Interface (UEFI) settings among the firmware update packages on IBM Fix Central.

Other settings are determined by the cluster's architecture and its intended use. Performance expectations are tricky because of the gap between theoretical peek results and practical achievements. But what is an allowed variation and where is the boundary to a faulty condition?

Health as a bottom-up approach

There are different levels of verification, from basic hardware examination to user application performance. These different stages can be grouped in three major categories:

Environment	Consists of all hardware-related items.
Operation	Covers all runtime parameters.
Performance	Summarizes the system's behavior under load condition.

Our experience from different cluster deployments has taught us that a healthy cluster is built from the bottom to the top. Therefore, you first must make sure that each single device works as expected before performing the next step. Not until all hardware and software have been checked is it worth running a benchmark job.

This approach leads to the pyramid model of verification shown in Figure 3-2 on page 20. Each level of verification depends on the correctness of the underlying stages. The higher in the pyramid you are before a problem is found, the more difficult it is to attribute it to a more foundational problem.

The following practices can help with the pyramid verification approach shown in Figure 3-2:

- For environment, ensure uniformity:
  - Verify attributes of hardware, such as type, model, and firmware level.
  - Verify hardware-oriented settings, such as hardware management controller and UEFI.
  - Verify hardware status, such as temperature, port state, memory, and PCI bus speed.
  - Verify infrastructure, such as InfiniBand topology, correct cabling, line speed, width, and error rate.
- For operation, validate consistency:
  - Verify mandatory resources, such as network connectivity and file systems.
  - Verify software parameters, such as kernel settings, available memory, required configurations, error rates, correct speeds, and widths.
- For performance, evaluate expectations:
  - Verify expected performance, such as throughput, usable memory or latency.

During the verification phase, the component in question is supposed to be isolated. That enables you to run intrusive tests. These tests use essential, and are inapplicable if other jobs share the same resource. In one case, the test might harm the other job. Alternatively, the result of the test might be inaccurate.

An example is measuring memory bandwidth on a compute node while a user job is running on the same node. The measurement results are misleading, and the user job might fail. For this reason, all performance tests are considered intrusive and must be handled with care.

The perception of health depends on yielding expected results. Thus, conduct both intrusive and passive tests.

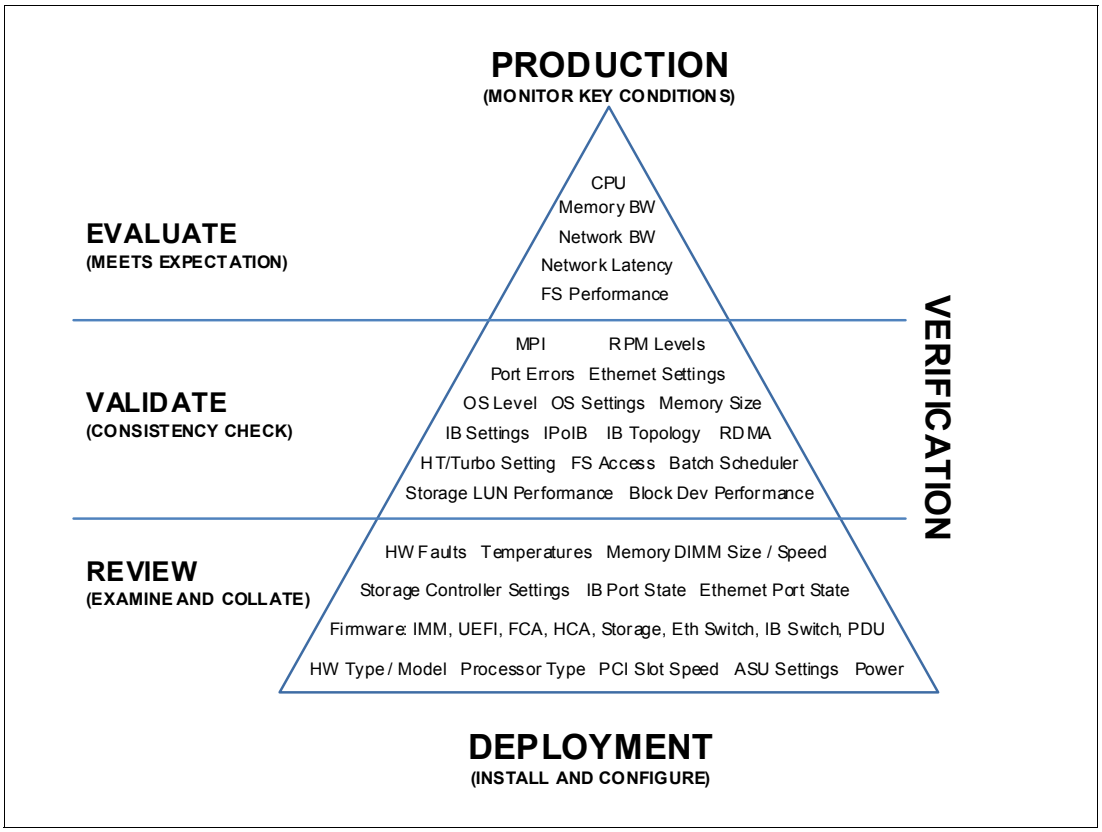

Figure 3-2 Verification pyramid

Burn-in

One method to reduce the probability of hardware failures before the bring-up operation is a burn-in cycle. For a limited time, the cluster environment is placed in a high workload condition so that all of the devices warm up to operational temperature, and all of the electrical and mechanical components are stressed. That provokes early life failures (or so-called infant mortality) of the hardware, and helps to identify loose cable connectors, insufficient electrical power, open fuses, or thermal issues.

Figure 3-3 shows a widely used model of failure probability in a product's lifetime, known as *the bathtub curve*. The challenge is to define the optimal time for a burn-in phase. If it is too short, there will still be a high number of faults after you start into production. Alternatively, a too-long stress condition causes faster aging, and it is a waste of energy. It is important to note that the model refers only to hardware faults. Software stability is not vulnerable to mechanical erosion or heat, but to many other age-independent factors.

Tip: A good practice is to chart failure rates. You should see a marked increase followed by a drop-off to a much lower and more steady rate, which is an indicator that you have exited the early-life failure cycle. Because of the statistical nature of failure rates, you will find that this technique works much better for large clusters than for smaller clusters.

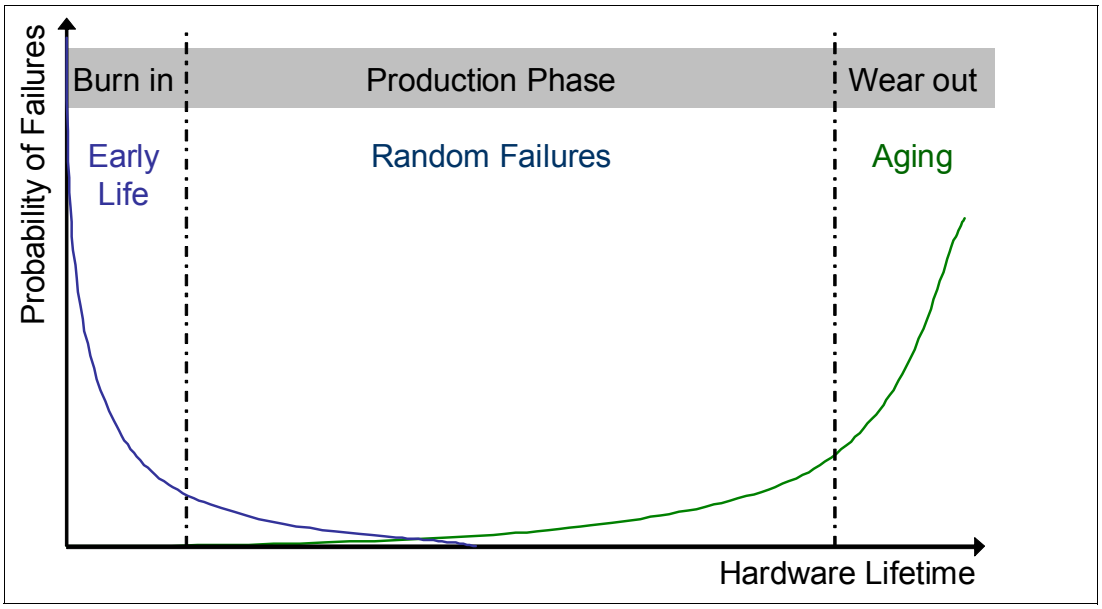

Figure 3-3 Example of a bathtub curve

3.3.3 Production phase (monitoring)

The following list includes suggestions for the production phase:

- Monitoring, keeping log files, correlating events
- Conducting passive tests until tests show an anomaly (monitoring)
- Interval checking (effect on performance)
- Spot checking (sacrificing performance for sanity)

4

Cluster components reference model

This chapter describes the High Performance Computing (HPC) cluster environment used by the team for this IBM Redbooks publication. In this chapter, we provide a detailed overview of the installed IBM hardware and software, and describe the configuration settings.

The following topics are described in this chapter:

- Overview of installed cluster systems
- ClusterA nodes hardware description
- ClusterA software description
- ClusterB nodes hardware description
- ClusterB software description
- ClusterC nodes hardware description
- ClusterC software description
- Interconnect infrastructure
- GPFS cluster installation in ClusterB

4.1 Overview of installed cluster systems

The lab environment used for this IBM Redbooks publication consists of three clusters with different architecture and hardware to perform the various system tests. These clusters are referred to as ClusterA, ClusterB, and ClusterC throughout this IBM Redbooks publication.

All Clusters are based on Red Hat Enterprise Linux (RHEL) release 6.3. The following systems are used:

- ClusterA is based on IBM System x iDataPlex® with eight compute nodes.
- ClusterB is based on IBM System x x3650 with six compute nodes.
- ClusterC is based on IBM Power Systems 740 with four compute nodes.

All nodes are installed using RHEL Server release 6.3 with kernel version 2.6.32-279.el6.x86_64 (ClusterB) and 2.6.32-279.el6.ppc64 (ClusterC). The only exception is the management server for ClusterA, which has RHEL Server release 6.4 installed, but is booted with the 6.3 kernel 2.6.32.279 as well.

All management servers are running the IBM Extreme Cloud Administration Toolkit (xCAT) software. The xCAT software is an open-source tool for highly scalable management and deployment of clusters.

More detailed information about xCAT can be found on the xCAT wiki website:

http://sourceforge.net/apps/mediawiki/xcat/index.php?title=Main_Page

You can also find more information about xCAT at the following IBM website:

http://www-03.ibm.com/systems/software/xcat/index.html

Each of the clusters is managed by one xCAT server. The details of the xCAT setup are not explained in this publication. The cluster nodes are equipped with local hard disk drives (HDDs), and are deployed as stateful nodes by the xCAT management nodes.

The clusters consist only of compute nodes and one management node. There are no special utility nodes, such as login nodes or scheduler nodes.

No external storage subsystems are available in the lab environment. Therefore, an IBM General Parallel File System (GPFS) cluster has been defined only on ClusterB, which provides a parallel file system for test purposes. This file system is defined on local disks built in two Network Shared Disk (NSD) servers.

More detailed information about GPFS can be found on the following website:

http://www-03.ibm.com/systems/software/gpfs/

You can also find more information about GPFS on the developerWorks website:

https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20%28GPFS%29

All of the cluster compute nodes are connected by an Ethernet and an InfiniBand infrastructure, which is actually physically shared to some degree between the three clusters and even more systems in the lab. The other systems are not part of this IBM Redbooks publication.

4.2 ClusterA nodes hardware description

ClusterA consists of eight IBM iDataPlex dx360 M4 nodes. The xCAT management node is installed. Table 4-1 provides a short overview of the components.

Table 4-1 Node hardware overview for ClusterA

Node	Model	Memory (gigabytes)	Processor
computeA1	IBM iDataPlex dx360 M4 7912-AC1	32 (16x2 GB)	E5-2680 0 @ 2.70 GHz Two packages
computeA2	IBM iDataPlex dx360 M4 7912-AC1	32 (16x2 GB)	E5-2680 0 @ 2.70 GHz Two packages
computeA3	IBM iDataPlex dx360 M4 7912-AC1	32 (16x2 GB)	E5-2680 0 @ 2.70 GHz Two packages
computeA4	IBM iDataPlex dx360 M4 7912-AC1	32 (16x2 GB)	E5-2680 0 @ 2.70 GHz Two packages
computeA5	IBM iDataPlex dx360 M4 7912-AC1	32 (16x2 GB)	E5-2680 0 @ 2.70 GHz Two packages
computeA6	IBM iDataPlex dx360 M4 7912-AC1	32 (16x2 GB)	E5-2680 0 @ 2.70 GHz Two packages
computeA7	IBM iDataPlex dx360 M4 7912-AC1	28 (14x2 GB) 2 DIMMs are missing	E5-2680 0 @ 2.70 GHz Two packages
computeA8	IBM iDataPlex dx360 M4 7912-AC1	32 (16x2 GB)	E5-2680 0 @ 2.70 GHz Two packages
xcatA1	IBM x3550 M4 7042-CR7	32 (16x2 GB)	E5-2640 0 @ 2.50 GHz One package

The compute nodes have two processor packages installed with eight cores each. Hyperthreading is enabled, therefore resulting in 32 processors made available to the operating system.

The management node has one processor package installed with six cores. Hyperthreading is disabled.

The InfiniBand attachment is 4X Fourteen Data Rate (FDR10) using host channel adapters (HCAs) of type Mellanox Technologies MT27500 Family [ConnectX-3] at firmware level 2.11.1260. The boards have product-set identification (PSID) IBM0F40110019. The xcatA1 management node does not have a connection to the InfiniBand fabric.

Important: In the computeA7 node, two dual inline memory modules (DIMMs) are missing. For example, this shows up in performance tests such as **daxpy**. See “Example of daxpy” on page 98.

4.3 ClusterA software description

xCAT version 2.8.2 is installed on the xcatA1 management node. This xCAT server is responsible for the management of other nodes as well which are not part of ClusterA.

The InfiniBand software stack on the compute nodes is Mellanox OpenFabrics Enterprise Distribution (OFED) 2.0-2.0.5. The fabric management is done by the Subnet Manager Mellanox UFM 4.0.0 build 19. The Unified Fabric Manager (UFM) software is installed as a single instance in non-high availability (HA) mode.

For parallel code execution, the IBM Parallel Environment (PE) 1.3.0.6-s006a is installed on all compute nodes. This software basically is an implementation of Message Passing Interface (MPI) by IBM.

For more information about IBM PE see the following website:

http://www-03.ibm.com/systems/software/parallel/

4.4 ClusterB nodes hardware description

This section describes the servers installed in ClusterB. ClusterB consists of four IBM x3650 M2 and two IBM x3650 M3 compute nodes, which also differ in their installed memory.

Table 4-2 provides a short overview of the components.

Table 4-2 Node hardware overview for ClusterB

Node	Model	Memory (GB)	Processor
computeB1	IBM x3650 M2 7947-AC1	32 (16x2 GB)	Intel Xeon X5560 @ 2.80 GHz Two packages
computeB2	IBM x3650 M2 7947-AC1	32 (16x2 GB)	X5560 @ 2.80 GHz Two packages
computeB3	IBM x3650 M2 7947-AC1	32 (16x2 GB)	X5560 @ 2.80 GHz Two packages
computeB4	IBM x3650 M2 7947-AC1	32 (16x2 GB)	X5560 @ 2.80 GHz Two packages
computeB5	IBM x3650 M3 7945-AC1	68 (17x4 GB) 1 DIMM is missing	X5680 @ 3.33 GHz Two packages
computeB6	IBM x3650 M3 7945-AC1	72 (18x4 GB)	X5680 @ 3.33 GHz Two packages
xcatB1	IBM x3650 M3 7945-AC1	32 (16x2 GB)	E5630 @ 2.53 GHz One package

The compute nodes have two processor packages installed with four cores (X5680) or six cores (X5680) each. Hyperthreading is enabled, therefore resulting in 16 (X5560) and 24 (X5680) processors made available to the operating system.

The management node has one processor package installed with four cores. Hyperthreading is disabled.

The InfiniBand attachment is 4X Quad Data Rate (QDR) using HCAs of type Mellanox Technologies MT26428 [ConnectX VPI PCIe 2.0 5GT/s - IB QDR / 10GigE] at firmware level 2.9.1000. The system boards have PSID MT_0D81120009, except for computeB2 which has PSID MT_0BB0120003 installed.

The xcatB1 management node does not have a connection to the InfiniBand fabric.

Note that in the computeB5 node, 14 GB DIMM is missing.

4.5 ClusterB software description

xCAT Version 2.8.1 is installed on the xcatB1 management node. This xCAT server is responsible for other nodes as well.

The InfiniBand software stack on the compute nodes is Mellanox OFED 1.5.3-4.0.22.3. The fabric management is done by opensm-4.0.0 as provided by Mellanox OFED.

For parallel code execution, the MPI version IBM PE 1.3.0.6-s006a is installed on all compute nodes.

In addition, a GPFS cluster with GPFS Version 3.5.0-9 is installed on the compute nodes. More details about the GPFS setup are described in section 4.9, “GPFS cluster” on page 33.

4.6 ClusterC nodes hardware description

This section describes the servers installed in ClusterC. ClusterC consists of four IBM Power 740 compute nodes. Table 4-3 provides a short overview of the components.

Table 4-3 Node hardware overview for ClusterC

Node	Model	Memory (GB)	Processor
computeC1	IBM Power 740 8205-E6B	128	IBM POWER7® @3550 MHz 16 processors
computeC2	IBM Power 740 8205-E6B	128	POWER7 @3550 MHz 16 processors
computeC3	IBM Power 740 8205-E6B	256	POWER7 @3550 MHz 16 processors
computeC4	IBM Power 740 8205-E6B	256	POWER7 @3550 MHz 16CPUs
xcatC1	IBM Power 750 8233-E8B	32	POWER7 @3300 MHz 8 processors

The compute nodes have 16 processors installed. Simultaneous multithreading SMT4 is enabled, therefore resulting in 64 processors made available to the operating system.

The management node has 8 processors installed. Simultaneous multithreading SMT4 is enabled, resulting in 32 processors made available to the operating system.

The InfiniBand attachment is 4X QDR using HCAs of type Mellanox Technologies MT26428 [ConnectX VPI PCIe 2.0 5 GT/s - IB QDR / 10 GigE] (rev b0).

Nodes computeC1 and computeC2 have two adapters installed at firmware level 2.9.1000.

Nodes computeC3 and computeC4 have one adapter installed at firmware level 2.9.1316.

All system boards have PSID IBM0FA0000009, except for computeC1, which has PSID MT_0D80120009 installed.

The xcatC1 management node does not have a connection to the InfiniBand fabric.

4.7 ClusterC software description

xCAT Version 2.8.2 is installed on the xcatC1 management node. This xCAT server is responsible for other nodes as well.

The InfiniBand software stack on the compute nodes is Mellanox OFED 2.0-2.0.5. The fabric management is done by opensm-4.0.0 as provided by Mellanox OFED.

For parallel code execution, the MPI version IBM PE 1.3.0.5-s005a is installed on compute nodes computeC1 and computeC2. IBM PE 1.3.0.7-s007a is installed on computeC3 and computeC4.

4.8 Interconnect infrastructure

The cluster components are connected by Ethernet and InfiniBand. InfiniBand is used for native InfiniBand traffic and IP over InfiniBand (IPoIB). This section outlines the details from an InfiniBand and IP view.

4.8.1 InfiniBand

The nodes for all of the clusters are connected to a single InfiniBand switch. In terms of the InfiniBand topology, each of the clusters can therefore be called a *fat-free topology*. For ClusterB and ClusterC, which are connected to a single 36-port switch, this is a non-blocking configuration.

The compute nodes in ClusterA are connected to a modular InfiniBand switch of type Mellanox SX6512 216-Port InfiniBand Director Switch. Note that the ClusterA nodes are distributed across two leaf boards within the modular switch, as shown in Figure 4-1. Therefore, depending on the InfiniBand routes that are calculated by the Subnet Manager, this configuration is not necessarily non-blocking.

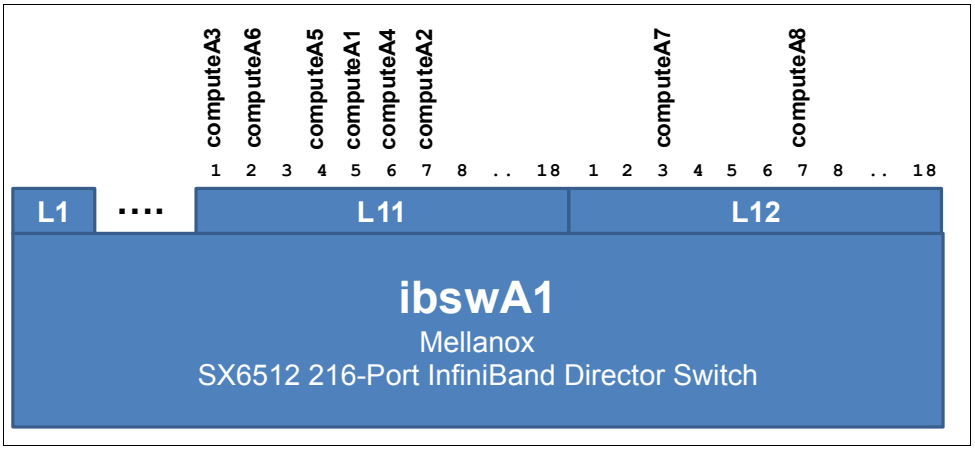

Figure 4-1 InfiniBand connections for ClusterA

The connections in ClusterA are 4X FDR10 for all nodes, supporting a rate of 40 Gbps. The cluster is run using InfiniBand maximum transmission unit (MTU) size 4096.

The compute nodes in ClusterB and ClusterC are connected to a 36-port InfiniBand switch of type Mellanox SX3036.

Figure 4-2 shows the InfiniBand connections of the compute nodes in ClusterB. All six nodes are connected to ports on one single switch.

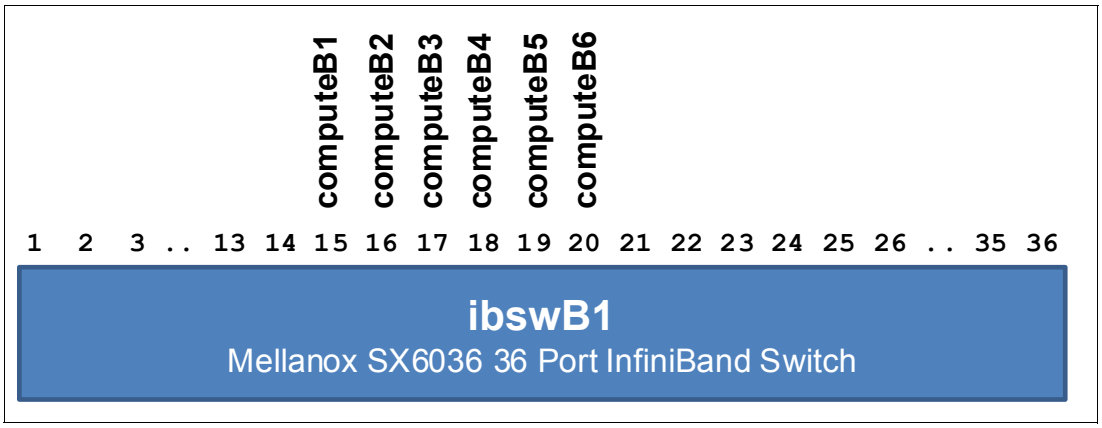

Figure 4-2 InfiniBand connections for ClusterB

Figure 4-3 shows the InfiniBand connections of the compute nodes in ClusterC. All four nodes are connected to ports on one single switch.

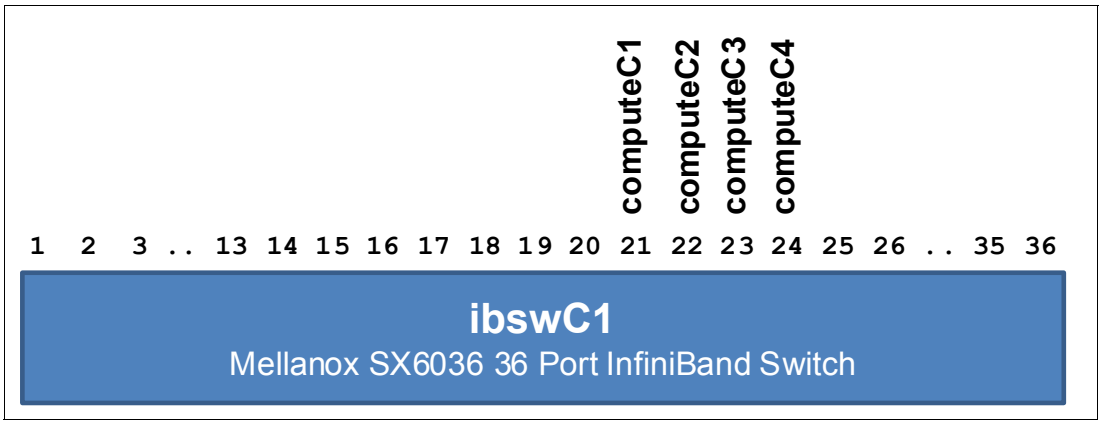

Figure 4-3 InfiniBand connections for ClusterC

The connections in ClusterB and ClusterC are 4X QDR for all nodes, supporting a rate of 40 Gbps. The effective data rate that can be achieved is limited to 32 Gbps.

Note: ibswB1 and ibswC1are actually the same physical machine that is shared between these clusters. For the sake of simple naming, the real names of components are translated to the respective cluster context in which they are used.

ClusterB nodes are running with InfiniBand MTU size 2048, and ClusterC nodes are running with InfiniBand MTU size 4096.

Subnet manager

ClusterA has a subnet manager based on Mellanox UFM, which is running on a separate node in the Fabric, smA1.

There are no configuration options changed from the default values.

ClusterB and ClusterC are running the `opensmd` daemon as subnet manager on an IBM BladeCenter® node that is connected to the fabric. Because the fabric is shared between both clusters, there is actually one single subnet manager running, shared by both clusters. For the sake of simplicity, this single server is named smB1 or smC1, depending on the cluster context that is being described.

There are no configuration options changed from the default values.

4.8.2 Ethernet Infrastructure

The underlying physical Ethernet infrastructure with details about switches and their configuration is not subject to this IBM Redbooks publication. All components are connected to a lab-wide network setup shared by many other components, and documentation of any details about that is out of scope. For the Ethernet view, it is sufficient to describe the IP layer. See 4.8.3, “IP Infrastructure” on page 30 for details.

4.8.3 IP Infrastructure

In terms of IP networking, the nodes are interconnected by different subnetworks. These networks are based on Ethernet and InfiniBand using the IPoIB protocol.

Table 4-4 shows the IP subnetworks used by the clusters, with a short description.

Table 4-4 IP subnet overview

IP subnet	Cluster	Description
10.0.0.0/8	A,B, and C (B and C on the same L2)	Base network for management and provisioning by xCAT
20.0.0.0/8	A and B	IPoIB network for ib0 interfaces in ClusterA and ClusterB
21.0.0.0/8	A	IPoIB network for ib1 interfaces in ClusterA
50.0.0.0/8	A and B	Management VLAN
60.0.0.0/8	A, B, and C	Network for login to the xCAT servers from outside
70.0.0.0/8	C	IPoIB network for ib0 interfaces in ClusterC
71.0.0.0/8	C	IPoIB network for ib1 interfaces in ClusterC

IP subnet 10.0.0.0/8 is the base network to which all nodes in all clusters have connections. This network is the xCAT internal provisioning network. ClusterB and ClusterC share the same layer2 network infrastructure. ClusterA runs this subnet in a separate layer2 network.

IP subnets 20.0.0.0/8 and 21.0.0.0/8 are IPoIB subnetworks used by ClusterA and ClusterB. Any ib0 interfaces are in subnet 20.0.0.0/8. If configured, any ib1 interfaces are placed into subnet 21.0.0.0/8.

IP subnet 50.0.0.0/8 actually is used by two clusters, ClusterA and ClusterB only, but is separated on layer 2 by employing different VLANs. Only the xCAT management nodes of ClusterA and ClusterB are connected to this network, using tagged VLAN interfaces.

IP subnet 60.0.0.0/8 is the network from which users access the cluster management nodes from within the lab infrastructure. This is VLAN255. Note that the compute nodes are also connected to this VLAN.

IP subnets 70.0.0.0/8 and 71.0.0.0/8 are IPoIB subnetworks used by ClusterC. Any ib0 interfaces are in 70.0.0.0/8. If configured, ib1 interfaces are placed into 71.0.0.0/8.

The following figurers show the clusters' interconnections from the IP view.

The IP network view for ClusterA is shown in Figure 4-4.

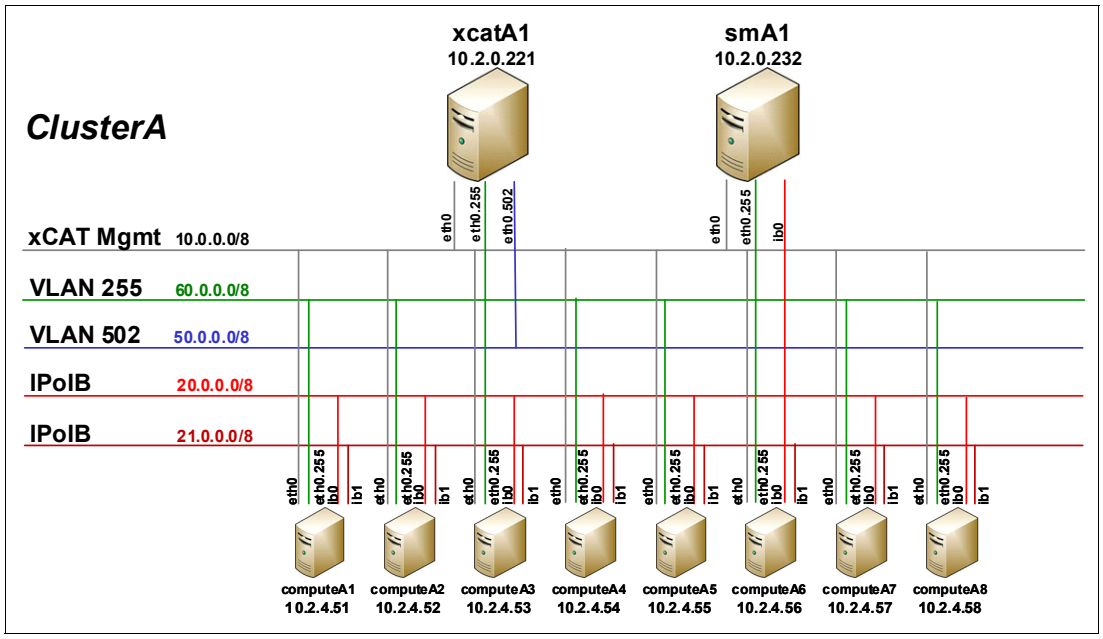

Figure 4-4 IP network view of ClusterA

The IP network view for ClusterB is shown in Figure 4-5.

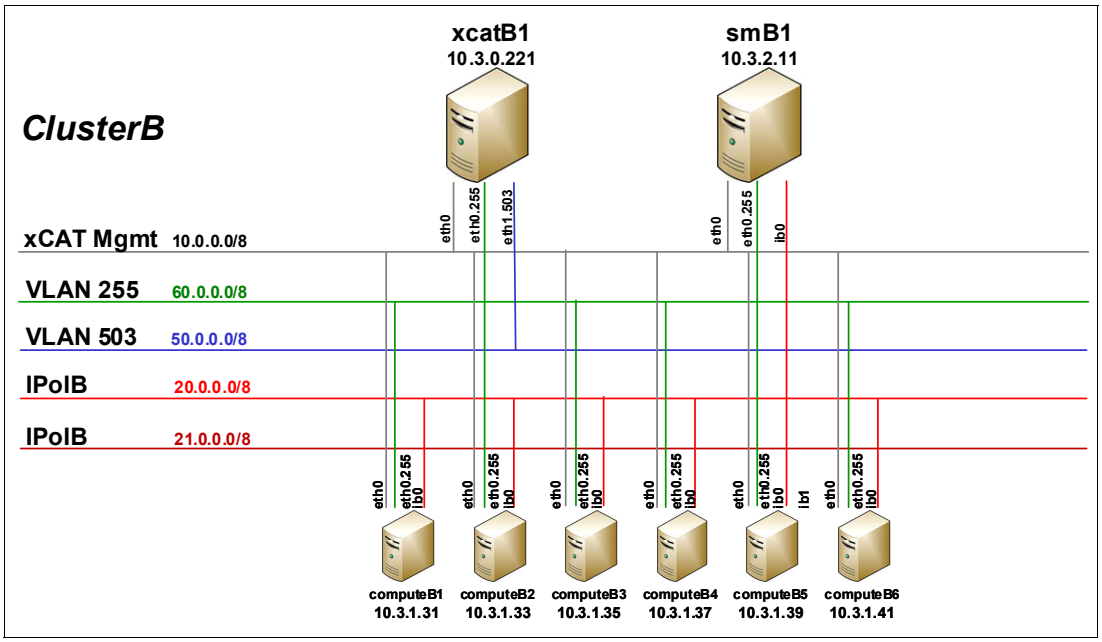

Figure 4-5 IP network view for ClusterB

Exception: VLAN502 is not used in this cluster. Instead, VLAN503 is used for 50.0.0.0/8.

The IP network view for ClusterC is shown in Figure 4-6.

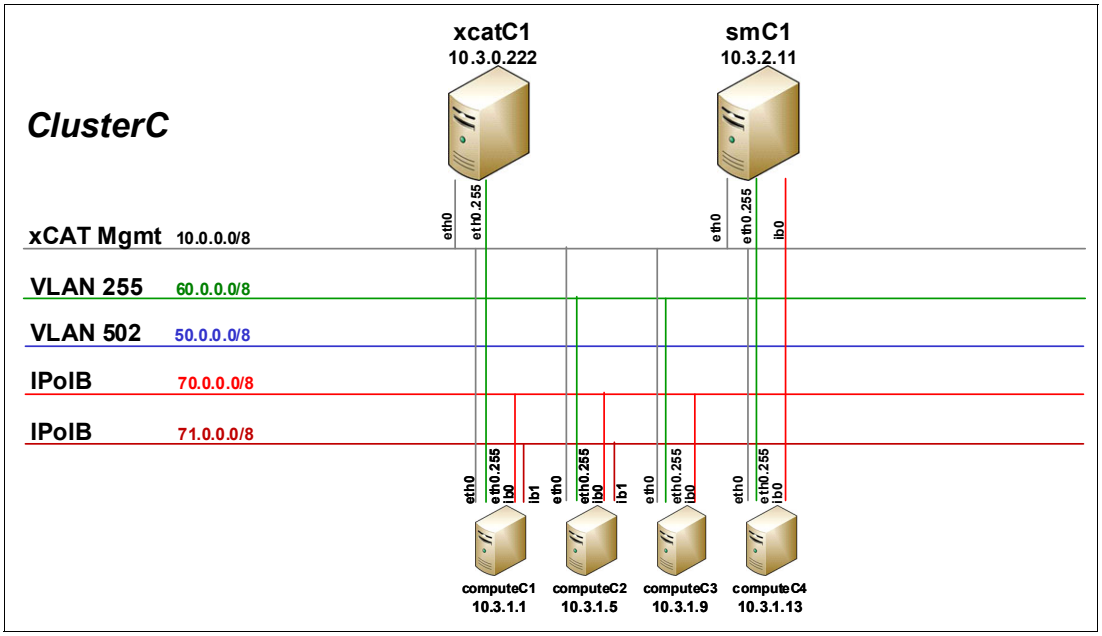

Figure 4-6 IP network view for ClusterC

Note: smC1 has the same IP as smB1. This is because it is actually the same physical machine that is shared between these clusters. For the sake of simple naming, the real names of the servers have been changed to provide easier reading.

Also note that two of the servers, computeC1 and computeC2 have a second IPoIB interface, ib1.

4.9 GPFS cluster

A GPFS cluster using software version 3.5.0.9 is installed on the ClusterB compute nodes. There is no external storage subsystem available on any of the clusters. Two nodes in ClusterB have therefore been configured to export their local disk `/dev/sdb` as an NSD into the GPFS file system.

The GPFS cluster is defined as shown in Example 4-1. The cluster has been defined in the 10.0.0.0/8 subnet.

Example 4-1 GPFS cluster overview

```
[root@computeB5 ~]# /usr/lpp/mmfs/bin/mmlscluster

GPFS cluster information
========================
  GPFS cluster name:         gpfsB.computeB5
  GPFS cluster id:           9191008596370922699
  GPFS UID domain:           gpfsB.computeB5
  Remote shell command:      /usr/bin/ssh
  Remote file copy command:  /usr/bin/scp

GPFS cluster configuration servers:
-----------------------------------
  Primary server:    computeB5
  Secondary server:  computeB6

 Node  Daemon node name  IP address  Admin node name  Designation
-----------------------------------------------------------------
   1   computeB1        10.3.1.31   computeB1
   2   computeB2        10.3.1.33   computeB2
   3   computeB3        10.3.1.35   computeB3
   4   computeB4        10.3.1.37   computeB4       quorum
   5   computeB5        10.3.1.39   computeB5       quorum-manager
   6   computeB6        10.3.1.41   computeB6       quorum-manager

[root@computeB5 ~]#
```

The GPFS cluster configuration settings are shown in Example 4-2. No special settings have been applied to the configuration, except for enabling RDMA support, which is disabled by default. This is done by configuring `verbsRdma enable` and `verbsPorts mlx4_0/1` to define the InfiniBand port to be used for communication.

Example 4-2 Cluster configuration settings

```
[root@computeB5 ~]# /usr/lpp/mmfs/bin/mmlsconfig
Configuration data for cluster gpfsB.computeB5:
-----------------------------------------------
myNodeConfigNumber 5
clusterName gpfsB.computeB5
clusterId 9191008596370922699
autoload no
dmapiFileHandleSize 32
minReleaseLevel 3.5.0.7
verbsRdma enable
verbsPorts mlx4_0/1
adminMode central

File systems in cluster gpfsB.computeB5:
----------------------------------------
/dev/fsB
[root@computeB5 ~]#
```

One file system consisting of two local server disks has been created, because no external storage system is available. Each NSD is handled by one of the NSD servers, computeB5 and computeB6. The NSDs are defined as shown in Example 4-3.

Example 4-3 NSD definitions for GPFS cluster gpfsB.computeB5

```
[root@computeB5 ~]# /usr/lpp/mmfs/bin/mmlsnsd -M

 Disk name    NSD volume ID      Device         Node name                Remarks
---------------------------------------------------------------------------------
-----
 nsdB5        0A03012752406843   /dev/sdb       computeB5               server
node
 nsdB6        0A03012952402347   /dev/sdb       computeB6               server
node

[root@computeB5 ~]#
```

These NSDs are put in the file system named `fsB`. This file system has been created with the settings shown in Example 4-4.

Example 4-4 File system settings for GPFS file system fsB

```
[root@computeB5 ~]# /usr/lpp/mmfs/bin/mmlsfs fsB
flag                value                    description
------------------- ------------------------ -----------------------------------
 -f                 8192                     Minimum fragment size in bytes
 -i                 512                      Inode size in bytes
 -I                 16384                    Indirect block size in bytes
 -m                 1                        Default number of metadata replicas
 -M                 1                        Maximum number of metadata replicas
 -r                 1                        Default number of data replicas
 -R                 1                        Maximum number of data replicas
 -j                 cluster                  Block allocation type
 -D                 nfs4                     File locking semantics in effect
 -k                 nfs4                     ACL semantics in effect
 -n                 10                       Estimated number of nodes that will
mount file system
 -B                 262144                   Block size
 -Q                 user;group;fileset       Quotas enforced
                    none                     Default quotas enabled
 --filesetdf        No                       Fileset df enabled?
 -V                 13.23 (3.5.0.7)          File system version
 --create-time      Mon Sep 23 12:23:50 2013 File system creation time
 -u                 Yes                      Support for large LUNs?
 -z                 No                       Is DMAPI enabled?
 -L                 4194304                  Logfile size
 -E                 Yes                      Exact mtime mount option
 -S                 No                       Suppress atime mount option
 -K                 whenpossible             Strict replica allocation option
 --fastea           Yes                      Fast external attributes enabled?
 --inode-limit      140288                   Maximum number of inodes
 -P                 system                   Disk storage pools in file system
 -d                 nsdB5;nsdB6              Disks in file system
 --perfileset-quota no                       Per-fileset quota enforcement
 -A                 no                       Automatic mount option
 -o                 none                     Additional mount options
 -T                 /fsB                     Default mount point
 --mount-priority   0                        Mount priority
[root@computeB5 ~]#
```

There are no special configuration options set for the file system. Note that the `inode-limit` has not been specified explicitly, so there is a low limit calculated by default.

5

Toolkits for verifying health (individual diagnostics)

To determine the health of a cluster, it is necessary to get the current state of all of the components that build it up in most installations:

- Compute nodes
- Ethernet network
- InfiniBand network
- Storage

In this chapter, we describe the IBM Cluster Health Check (CHC) toolkit, which is used to perform checks on these components. Working with these results, we are able to state if the cluster is healthy.

This chapter provides information about the following topics:

- Introduction to CHC
- Tool output processing methods
- Compute node
- Ethernet network: Port status, speed, bandwidth, and port errors
- InfiniBand: Port status, speed, bandwidth, port errors, and subnet manager
- File system: Accessibility, usage, and read/write performance

5.1 Introduction to CHC

The CHC framework provides an environment for integrating and running a subset of individual checks, which can be performed on a single node or a group of nodes. The tests are categorized into different tool types and groups, so that only checks for a specific part of the cluster can be performed. Because the framework and toolset is extensible, users can also create their own groups and tools for checking different components of the cluster.

The primary wrapper tool is **hcrun**, which provides access to all tools and passes them the environment.

5.1.1 Requirements

The CHC toolkit should be installed on the IBM Extreme Cloud Administration Toolkit (xCAT) server, because many of the individual checks and the framework use xCAT commands. In addition, some of the health checks are Message Passing Interface (MPI) parallel applications that run using Parallel Operating Environment (POE) runtime environment.

The following list includes the required software packages:

- xCAT
- POE environment
- Python 2.6

Note: The IBM CHC tools can be downloaded from the IBM High Performance Computing (HPC) Central website:

https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Welcome%20to%20High%20Performance%20Computing%20%28HPC%29%20Central/page/IBM%20Cluster%20Health%20Check

5.1.2 Installation

The CHC toolkit is delivered as a Red Hat Package Manager (RPM) package, and is installed as user root with the **rpm** command:

```
# rpm -ihv cluster-healthcheck-1.1.0.0-0.noarch.rpm
```

The **rpm** command installs to `/opt/ibmchc`.

To run the tool directly from the command line, there are two possibilities:

- Edit `~/.profile` and add the following line:

  ```
  # export PATH=$PATH:/opt/ibmchc/bin
  ```

- Create a symbolic link (symlink) to `/opt/ibmchc/bin/hcrun` with the following command:

  ```
  # ln -s /opt/ibmchc/bin/hcrun /usr/bin/hcrun
  ```

5.1.3 Configuration

The CHC and verification tools are developed to check the health of individual cluster components, or a set of cluster components. There are three different types of configuration files: global, tool, and group.

The general format of the configuration file is keyword value pairs defined under sections. The global settings of the suite can be changed in `/etc/opt/ibmchc/chc.conf`, as shown in Example 5-1.

Example 5-1 The /etc/opt/ibmchc/chc.conf file

```
[MasterConfig]

# The list of high-level directories under
# which configuration of different health
# checking tools and groups are defined.
configpath =
/opt/ibmchc/tools:/opt/ibmchc/ppc64/tools:/opt/ibmchc/x86_64/tools:/opt/ibmchc/conf

# stdout results of health checking tools
toolsoutput = /var/opt/ibmchc/log/toolsoutput

# The summary of health checking tools
# is logged in this file
summaryfile = /var/opt/ibmchc/log/hcrun_sum.log

# The hcrun logs are stored in this file.
hcrunlog = /var/opt/ibmchc/log/hcrun.log

# The user id which is used to run MPI jobs
# Tools can access user name using env variable $HC_USER
# No default user
hcuser =

# The comma-separated UFM server host names or ip
# addresses. No default values.  @01
# Tools can access subnet Managers using env variable $HC_SUBNET_MANAGERS

subnet_managers =

# The comma-separated IB Switch node groups
# Tools can access IB Swicth groups using env variable $HC_IB_SWITCHES
# No default values.

ib_switches =
```

The defaults are fine for most users. However, the following parameters need to be updated to match each unique cluster's environment:

- The **hcuser** parameter

  Some of the tests run with the IBM Parallel Environment (PE). These tests must not be submitted as root. Therefore, a dedicated user must be available. The value is exported through the **HC_USER** environment variable. The user must have authority to run applications in the PE environment.

- The **subnet_managers** parameter

  If an InfiniBand subnet manager is available, its host name can be specified in the configuration file. Specify multiple servers as a comma-separated list. The value is exported through the **HC_SUBNET_MANAGERS** environment variable. Tools that require access to subnet manager servers will use this environment variable.

- The **ib_switches** parameter

  Specify the xCAT node groups for the InfiniBand switches as a comma-separated list. Note that this requires the switches to be in the xCAT database and have Secure Shell (SSH) access enabled. Different groups for different models of switches should be created in the xCAT database. The value is exported through the **HC_IB_SWITCHES** environment variable. Tools that require access to InfiniBand switches will use this environment variable.

Tool configuration files

Each of the available tools has its own configuration file, as shown in Table 5-1. Each tool configuration needs to be somewhere in or under the path that is specified by the **configpath** parameter in the global configuration file. All of the tool configuration files must have the `.conf` extension. Table 5-1 shows all of the available parameters for the tool configuration file.

Table 5-1 Health check tool-specific configuration file

Parameter	Required	Value type	Remark
[HealthCheckTool]	Yes	n/a	Section header for the configuration file.
name	Yes	String	Name of the check.
description	Yes	Text	Short description of the tool.
type	Yes	String	Tool type to which this tool belongs. For display purposes only.
executable	No	Path to executable script or binary	Specifies the program to be run. **Default if not set**: Tool config path without `.conf` extension.
copy	No	`yes` | `no`	Specifies if the tool needs to be copied to the node or node group (uses **xdsh -e**) **Default if not set**: `yes`.
starttool	No	`all` | `local`	Specifies if the tool needs to be run on all nodes (provided in a node list to **hcrun**) or just on the node where **hcrun** is running. **Default if not set**: `all`.
arguments	No	String	Optional command-line arguments for the executable file. It is possible to specify non-default arguments in the **hcrun** command line after the tool name argument. If any arguments have multiple words separated with a space, then they need to be enclosed within double quotation marks.

Parameter	Required	Value type	Remark
environment	No	Tool names	Optional environment variables exported for the tool. Declare multiple variables delimited with a semicolon (;). Each environment variable is defined as variable name and value pairs separated by an equal sign (=). If **`hcrun -n <NODERANGE>`** is specified, the node range is exported as `HC_NODERANGE`.
precheck	No	Toolnames	Optional comma-separated list of toolnames that runs before this tool.
predatacollectiontool	No	Toolnames	Optional comma-separated list of toolnames that runs before this tool to collect data.
postdatacollectiontool	No	Toolnames	Optional comma-separated list of toolnames that runs after this tool to collect data.
processmethod	No	`xcoll | compare | xcoll,compare`	Optional comma-separated list of supported tool output processing methods shown in the processing method.
passexitcode	No	Numeric	Optional comma-separated list of exit codes that consider the test *passed*. The exit code `0` is considered the default pass exit code.
failexitcode	No	Numeric	Optional comma-separated list of exit codes that consider the test *failed*. The non-zero exit codes are considered default fail if no pass exit codes are defined. If either pass exit codes, warning exit codes, or error exit codes are defined, exit codes that are not members of those exit codes are considered fail exit codes.
warningexitcode	No	Numeric	Optional comma-separated list of exit codes that consider the test result a *warning*.
errorexitcode		Numeric	Optional comma-separated list of exit codes that consider the test result an *error*.

Example 5-2 shows that the executable file is copied to all of the specified nodes, runs on all of the nodes, and that only an exit code of zero (0) is considered to be a successful check. The `fs_usage` check is part of the tool type node. The `fs_usage` check supports both tool output processing methods (`xcoll` and `compare`), as shown in “Tool output processing methods” on page 44.

Example 5-2 The /opt/ibmchc/tools/node/fs_usage.conf file

```
[HealthCheckTool]
name=fs_usage
description=Checks the usage of the provided file systems
type=node
executable=/opt/ibmchc/tools/node/fs_usage
arguments=/tmp /var/tmp
copy=yes
starttool=all
processmethod=xcoll,compare
passexitcode=0
errorexitcode=128
```

The configuration file makes it easy to implement new checks into the framework of **`hcrun`**. Even the scripting language can be chosen easily, for example Bash, Python, or Perl script, if the interpreter is available on the target node.

There are two types of scripts:

- Simple query scripts that query values and output them
- Check scripts that have logic implemented and provide exit codes

Follow these steps to run a new script:

1. Create the check script.
2. Copy your script to the appropriate arch folder (`/opt/ibmchc/tools`, `/opt/ibmchc/ppc64/tools`, or `/opt/ibmchc/x86_64/tools`).
3. Create the configuration file for the script in the same folder, with a `.conf` extension.
4. Make the script executable.

The new tool shows up when **`hcrun -l`** is run.

Group configuration files

The group configuration file defines multiple tools that should run together in a specific order. There are two types of groups:

- Predefined groups that come with the RPM

  Predefined groups are in `/opt/ibmchc/conf/groups/chc`. These configuration files should not be edited.

- User-defined groups

  User-defined groups are in `/opt/ibmchc/conf/groups/user`.

Example 5-3 on page 43 shows a group configuration file example:

- All tools are run in the given order.
- The Section header is called `[HealthCheckToolsGroup]`.
- The `name` and `description` attributes are strings.
- `Tools` is a comma-separated list of toolnames in a specific order.

Example 5-3 The /opt/ibmchc/conf/groups/chc/node_check.conf file

```
[HealthCheckToolsGroup]
name=node_check
description=This group has node check tools
tools=cpu, memory, gpfs_state, leds, temp, nfs_mounts, firmware, lsf_state,
fs_usage, os
```

To overwrite tool keywords of the tool configuration files (such as `arguments` or `environment`), add the following line to the group configuration file, as shown in Example 5-4:

```
<toolname>.<keywordname>=<new value>
```

Example 5-4 Same as the previous example, but with overwritten fs_usage arguments

```
[HealthCheckToolsGroup]
name=node_check
description=This group has node check tools
tools=cpu, memory, gpfs_state, leds, temp, nfs_mounts, firmware, lsf_state,
fs_usage, os
fs_usage.arguments=/tmp
```

To create such a group configuration file, only one step is needed. You create the configuration file with extension `.conf` in `/opt/ibmchc/conf/groups/user`. The new group shows up when **`hcrun -l`** is run.

5.1.4 Usage

The primary program to run is in `/opt/ibmchc/bin`, and is called **`hcrun`**. The program should be run as a superuser. When called with the **`-h`** flag, a list of the supported options is displayed, as shown in Example 5-5.

Example 5-5 The hcrun supported options list

```
Usage: hcrun [options] [[ -s seed_node ] -m { xcoll | compare }] {
group[,group,...] | -t tool[,tool>... | tool_args] }

The interactive tool to check cluster health

Options:
  -h               show this help message and exit
  -l               List the configured Cluster Health Checking tools and
                   groups
  -v               Run health check and hcrun command in verbose mode
  -c               Run all health checks of a group even if one or more checks
                   fail
  -n NODERANGE     The node range on which the health checks would be run
  -s SEED_NODE     The seed node against which the query health check results
                   are compared
  -m PROCESS_MODE  The method ( xcoll | compare ) used to process the health
                   check results
  -p               Preview the order in which the tools of the specified
                   groups are ran
  -f               Force the execution of health checks on nodes ignoring
                   nodes can not be communicated
```

The **hcrun -l** command lists the available check tools, sorted by tool type and groups. All intrusive tools have a `run_` prefix. Intrusive tools should not be run when the nodes are in production. All tools are covered in detail in the following sections of this chapter.

5.2 Tool output processing methods

The CHC framework provides different methods to process the output of a tool. The current version provides the following modes:

- The `plain` (no post processing) mode
- The `xcoll` mode
- The `compare` mode

Some tools cannot support `compare` or `xcoll`, because the output of those tools might not be in a format suitable to use the `xcoll` and `compare` methods. Therefore, the tools must define supported methods, as shown in “Tool configuration files” on page 40. To run a tool with one of these methods, use the following syntax:

hcrun -m {xcoll|compare}

This works for groups as well. If a tool in a group does not support the specified method, the tool falls back to `plain` mode. A run without the **-m** flag runs the tool or group without further processing (`plain`).

5.2.1 The plain mode

The `plain` mode is the easiest of all three output processing methods. In `plain` mode, **hcrun** runs the tool and shows the plain output without changing anything about it. This is useful to check values of individual nodes. The following configuration keywords are used:

- Tool config keyword value `processmethod=‘’` (not defined)
- No **hcrun** option

5.2.2 The xcoll mode

In `xcoll` mode, **hcrun** runs the tool. Afterward, it pipes the output to the xCAT **xcoll** command, and **xcoll** summarizes identical output of different nodes to one *output group*. Example 5-6 shows the firmware tool with the `xcoll` method. The following configuration keywords are used:

- Tool config keyword value `processmethod=xcoll`
- The **hcrun** command **-m xcoll** option

For more information about xcoll, see the following web page:

http://xcat.sourceforge.net/man1/xcoll.1.html

Example 5-6 The firmware tool with the xcoll method

```
# ./hcrun -n redbook -m xcoll -t firmware
====================================
computeA1,computeA2,computeA4,computeA6,computeA7,computeA8
====================================
BMC Firmware: 3.45 (1AOO47P 2013/10/04 12:15:42)
UEFI Version: 1.10 (FHE105LUS  2013/10/17)
```

```
====================================
computeA3,computeA5
====================================
BMC Firmware: 3.45 (1AOO47P 2013/10/04 12:15:42)
UEFI Version: 1.00 (FHD102GHI 2013/03/12)
```

5.2.3 The compare (config_check) mode

The `compare` processing method uses `/opt/ibmchc/tools/config/config_check` to compare the output of multiple nodes for tools that should output the same values each time. The following configuration keywords are used:

- Tool config keyword value `processmethod=compare`
- The **`hcrun`** command **`-m compare`** option

The `config_check` process is a front end to the **`sinv`** xCAT command, which collects the output and creates a summary of the results. This check is not done automatically by **`sinv`**. For more information about **`sinv`**, see the following web page:

http://xcat.sourceforge.net/man1/sinv.1.html

The output from each node is compared with a baseline template[1]. The baseline can be created by the CHC framework on the initial run with the seed node variable. This creates a baseline template with the output of this node. The seed node value can be set with the **`$HC_SEED_NODE`** environment variable, or with the **`-s SEED_NODE`** command-line option of **`hcrun`**. On further invocations of the check, the results of the different nodes are checked against this baseline if no new seed node is specified.

Important: It is extremely important that the seed node has the expected results, so that subsequent invocations only reveal mismatches to a known good result.

In Example 5-7, a simplified version of the `cpu` check (only CPU model and governor) is run in `compare` mode with computeA1 as the baseline (seed node). The check fails because there are three different outputs. Note that computeA2 and computeA4 have the same output as computeA1. These nodes are fine, but computeA5 has different output. The computeA6 and computeA3 nodes have the same output, but it is different from the computeA output.

Example 5-7 First run with non-existing template, with computeA1 as the seed node

```
# ./hcrun -n redbook -s computeA1 -m compare -t cpu2

The following nodes match /opt/ibmchc/conf/templates/cpu2.template.config_check:
computeA2,computeA1,computeA4
The following nodes match /opt/ibmchc/conf/templates/cpu2.template.config_check_1:
computeA5
The following nodes match /opt/ibmchc/conf/templates/cpu2.template.config_check_2:
computeA6,computeA3

Info:Details can be found in file:
/var/opt/ibmchc/log/cpu2.template/config_check.log.20131218_184859
The health check tool  cpu2                              [ FAILED ]
```

[1] Templates are a sinv concept. The same results from a group of nodes are gathered into a single result called a template. If a node range has multiple results, multiple templates are created.

In Example 5-8, the log file shows the whole process. The `config_check` process creates a new template with the output of computeA1. Afterward, the outputs of all other nodes are compared with this template.

Example 5-8 Log file with baseline template generation

```
# cat /var/opt/ibmchc/log/cpu2.template/config_check.log.20131218_184859
- Create template in /opt/ibmchc/conf/templates/cpu2.template.config_check
  - Use node computeA1
Remove old template
Copy /opt/ibmchc/x86_64/tools/node/cpu2 to computeA1:/tmp
* Template Results in /opt/ibmchc/conf/templates/cpu2.template.config_check
Copy /opt/ibmchc/x86_64/tools/node/cpu2 to redbook:/tmp
Building Report.

Command Complete.
Check report in /var/opt/ibmchc/log/cpu2.template/cpu2.template.sinvout.20131218_184859.

-- Checking results --
Command started with following input.
xdsh cmd:xdsh redbook -v /tmp/cpu2 .
Template path:/opt/ibmchc/conf/templates/cpu2.template.config_check.
Template cnt:10.
Remove template:NO.
Output file:/var/opt/ibmchc/log/cpu2.template/cpu2.template.sinvout.20131218_184859.
Exactmatch:NO.
Ignorefirst:NO.
Seed node:None.
file:None.

The following nodes match /opt/ibmchc/conf/templates/cpu2.template.config_check:
computeA2,computeA1,computeA4
The following nodes match /opt/ibmchc/conf/templates/cpu2.template.config_check_1:
computeA5
The following nodes match /opt/ibmchc/conf/templates/cpu2.template.config_check_2:
computeA6,computeA3
============================================================
Comparison against baseline for computeA3

- CPU Model:            Intel(R) Xeon(R) CPU           X5560  @ 2.80GHz
?                                                 - -------
+ CPU Model:            Intel(R) Xeon(R) CPU           X5680  @ 3.33GHz
?                                                    ++++++++
Changed from computeA1
 to computeA3

====================
-
============================================================
Comparison against baseline for computeA5

- scaling_governor:     ondemand
?                       ^^^   ^
+ scaling_governor:     performance
?                     ++++ ^   ^^
Changed from computeA1
 to computeA5

====================
-
```

In Example 5-9, no new seed node is specified, and there are baseline templates from the previous run, so the CHC framework compares the output to the existing template.

Example 5-9 The same tool with existing templates

```
# ./hcrun -n redbook -m compare -t cpu2

The following nodes match /opt/ibmchc/conf/templates/cpu2.template.config_check:
computeA2,computeA1,computeA4
The following nodes match /opt/ibmchc/conf/templates/cpu2.template.config_check_1:
computeA5
The following nodes match /opt/ibmchc/conf/templates/cpu2.template.config_check_2:
computeA6,computeA3

Info:Details can be found in file: /var/opt/ibmchc/log/cpu2.template/config_check.log.20131218_193430

The health check tool  cpu2                              [ FAILED ]

# cat /var/opt/ibmchc/log/cpu2.template/config_check.log.20131218_193430
Copy /opt/ibmchc/x86_64/tools/node/cpu2 to redbook:/tmp
Building Report.

Command Complete.
Check report in /var/opt/ibmchc/log/cpu2.template/cpu2.template.sinvout.20131218_193430.

-- Checking results --
Command started with following input.
xdsh cmd:xdsh redbook -v /tmp/cpu2 .
Template path:/opt/ibmchc/conf/templates/cpu2.template.config_check.
Template cnt:10.
Remove template:NO.
Output file:/var/opt/ibmchc/log/cpu2.template/cpu2.template.sinvout.20131218_184859.
Exactmatch:NO.
Ignorefirst:NO.
Seed node:None.
file:None.

The following nodes match /opt/ibmchc/conf/templates/cpu2.template.config_check:
computeA2,computeA1,computeA4
The following nodes match /opt/ibmchc/conf/templates/cpu2.template.config_check_1:
computeA5
The following nodes match /opt/ibmchc/conf/templates/cpu2.template.config_check_2:
computeA6,computeA3
============================================================
Comparison against baseline for computeA3

- CPU Model:            Intel(R) Xeon(R) CPU           X5560  @ 2.80GHz
?                                                   - -------
+ CPU Model:            Intel(R) Xeon(R) CPU           X5680  @ 3.33GHz
?                                                     ++++++++
Changed from computeA1
 to computeA3
====================
-
============================================================
Comparison against baseline for computeA5

- scaling_governor:     ondemand
?                      ^^^   ^
+ scaling_governor:     performance
?                     ++++ ^   ^^
Changed from computeA1
 to computeA5
====================
-
```

Remember: After any maintenance that changes the results, a new template has to be generated.

5.3 Compute node

In this section, we look at the **hcrun** tools that analyze a compute node's CPU, memory, and hardware. Use the following syntax for all of the commands used in the following sections:

```
# hcrun -n <nodegroup> -t <check to run> [additional tool args]
```

The following command-line examples use the `plain` processing method (see "The plain mode" on page 44) to show the actual output of each tool. In practice, with a large number of nodes, run in `xcoll` or `comparison` mode:

```
# hcrun -n <nodegroup> -m <xcoll|compare> -t <check to run> [additional tool args]
```

Note: The following sections are not directly related to tool types or groups. Some of the compute node tools are x86_64 specific. The tool was not added to **$PATH** and a symlink was not created in the following examples.

5.3.1 The leds check

The `leds` check has the following characteristics:

Intrusive	This check is non-intrusive.
Supported processing method	Plain.
Purpose	Checks the fault LED of the given node.

This check requires access to the remote management facility (integrated management module (IMM), flexible service processor (FSP), or similar), as shown in Example 5-10.

Example 5-10 Fault LED active

```
# ./hcrun -n computeA1 -t leds
Running rvitals for noderange computeA1
computeA1: LED 0x0000 (Fault) active to indicate system error condition.
computeA1: LED 0012 (PS) active to indicate Sensor 0x71 (Power Supply 2) error.
rvitals FAIL!
```

In general, there should not be any fault LEDs active, as shown in Example 5-11. If there are any, the reason and effect must be considered.

Example 5-11 Fault LED is off

```
# ./hcrun -n computeA1 -t leds
Running rvitals for noderange computeA1
computeA1: No active error LEDs detected
rvitals  Good.
```

5.3.2 The cpu check

The `cpu` check has the following characteristics:

Intrusive	This check is non-intrusive.
Supported processing methods	Plain, xcoll, and compare.
Purpose	Collects CPU and governor information about a node.

Example 5-12 shows sample output of this tool. The output shows the CPU model, Turbo mode and Hyper threading status, number of cores and sockets, and the governor settings. Turbo mode is not enabled on this node. Different outputs per node might be intentional, or they might indicate a configuration issue.

Example 5-12 The cpu tool run

```
# ./hcrun -n computeA1 -t cpu
computeA1: CPU Model:               Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz
computeA1: Turbo HW:                Off
computeA1: Turbo Engaged:           No
computeA1: HyperThreading HW:       Enable
computeA1: Socket(s):               2
computeA1: Core(s) per socket:      12
computeA1: Active Cores:            48
computeA1: scaling_governor:        userspace
computeA1: scaling_max_freq:        2700000
computeA1: scaling_min_freq:        1200000
```

5.3.3 The memory check

The `memory` check has the following characteristics:

Intrusive	This check is non-intrusive.
Supported processing methods	Plain, xcoll, and compare.
Purpose	This tool returns the total memory available on the node, and details for each dual inline memory module (DIMM) module that is installed in the system. Empty DIMMs are shown as well.

Example 5-13 shows the total memory available to the kernel, the size of the installed DIMMs, and the speed and configured clock speed. DIMM 8 is empty. Different outputs per node might be intentional, or they might indicate a configuration issue.

Example 5-13 Memory check

```
# ./hcrun -n computeA1 -t memory
computeA1: Memory 56430 MB
computeA1
computeA1: "8192 MB","DIMM 1","Bank 1","1866 MHz","1867 MHz"
computeA1: "8192 MB","DIMM 2","Bank 2","1866 MHz","1867 MHz"
computeA1: "8192 MB","DIMM 3","Bank 3","1866 MHz","1867 MHz"
computeA1: "8192 MB","DIMM 4","Bank 4","1866 MHz","1867 MHz"
computeA1: "8192 MB","DIMM 5","Bank 5","1866 MHz","1867 MHz"
computeA1: "8192 MB","DIMM 6","Bank 6","1866 MHz","1867 MHz"
computeA1: "8192 MB","DIMM 7","Bank 7","1866 MHz","1867 MHz"
computeA1: "No Module Installed","DIMM 8","Bank 8","Unknown","Unknown"
```

5.3.4 The os check

The `os` check has the following characteristics:

Intrusive	This check is non-intrusive.
Supported processing methods	Plain, xcoll, and compare.
Purpose	This tool checks operating system and kernel version on a node.

Example 5-14 shows Red Hat Enterprise Linux (RHEL) Server 6.4 (Santiago) with Kernel 2.6.32 running. All nodes should run the same software versions.

Example 5-14 Operating system and kernel version

```
# ./hcrun -n computeA1 -t os
computeA1: Description:     Red Hat Enterprise Linux Server release 6.4 (Santiago)
computeA1: Release:         6.4
computeA1: Codename:        Santiago
computeA1: Kernel:          2.6.32-358.el6.x86_64
```

5.3.5 The firmware check

The `firmware` check has the following characteristics:

Intrusive	This check is non-intrusive.
Supported processing methods	Plain, xcoll, and compare.
Purpose	This tool checks the baseboard management controller (BMC) and Unified Extensible Firmware Interface (UEFI) firmware on a node.

This check requires access to the remote management facility (IMM, FSP, or similar). In Example 5-15, the BMC firmware is version 3.45 and UEFI is version 1.10. Different outputs per node might be intentional (firmware tests), or they might indicate a firmware update issue.

Example 5-15 Firmware versions

```
# ./hcrun -n computeA1 -t firmware
computeA1: BMC Firmware: 3.45 (1AOO47P 2013/10/04 12:15:42)
computeA1: UEFI Version: 1.10 (FHE105LUS  2013/10/17)
```

5.3.6 The temp check

The `temp` check has the following characteristics:

Intrusive	This check is non-intrusive.
Supported processing methods	Plain, xcoll, and compare.
Purpose	This tool checks all temperature sensors on a node.

This check requires access to the remote management facility (IMM, FSP, or similar).

Example 5-16 shows sample output. The number of sensors can vary per node.

Example 5-16 Temperature checking example output

```
# ./hcrun -n computeA1 -t temp
computeA1: Ambient Temp: 27 C (81 F)
computeA1: CPU1 VR Temp VCO: 50 C (122 F)
computeA1: DIMM AB Temp: 46 C (115 F)
computeA1: GPU Outlet Temp: N/A
computeA1: HDD Inlet Temp: 43 C (109 F)
computeA1: HDD Outlet Temp: N/A
computeA1: Mezz Card Temp: N/A
computeA1: PCH Temp: 44 C (111 F)
computeA1: PCI Riser 1 Temp: 32 C (90 F)
computeA1: PCI Riser 2 Temp: N/A
computeA1: PIB Ambient Temp: 37 C (99 F)
```

5.3.7 The run_daxpy check

The `run_daxpy` check has the following characteristics:

Intrusive	This check is *intrusive*.
Supported processing method	Plain.
Purpose	This tool checks the memory bandwidth on a compute node.

Further details about the underlying `daxpy` check can be found in Appendix A, “Commonly used tools” on page 75.

Example 5-17 uses default values and lets the `daxpy` test determine the expected performance rate. The example test failed because one node did not meet the expected result (see Example 5-18).

Example 5-17 The run_daxpy check failed

```
# ./hcrun -n computeA1+1,computeA7 -t run_daxpy
Error: daxpy did not achieve expected performance.
Info: daxpy output can be found in /var/opt/ibmchc/data/daxpy.out
The health check tool run_daxpy                                  [  FAILED  ]
```

Example 5-18 shows the created output file located under `/var/opt/ibmchc/data/daxpy.out`. There is a performance failure on computeA7. The computeA1 and computeA2 nodes met the expected values. Further analysis must be performed on computeA7 to find the reason.

Tip: Running `run_daxpy` with **`-v`**: **`# hcrun -n <noderange> -t run_daxpy -v`** prints the output file to the console, too. For an example of additional arguments, see Example 5-19 on page 52.

Example 5-18 The run_daxpy output file

```
computeA1: estimated EXPECTED_PERF 85836
computeA1: computeA1: Expected_DAXPY=85836  OMP_NUM_THREADS=24 perf (DAXPY BW)=
93885
computeA1: computeA1: PERFORMANCE_SUCCESS: 93885 GF;
computeA2: estimated EXPECTED_PERF 85836
```

```
computeA2: computeA2: Expected_DAXPY=85836  OMP_NUM_THREADS=24 perf (DAXPY BW)=
93788
computeA2: computeA2: PERFORMANCE_SUCCESS: 93788 GF;
computeA7: estimated EXPECTED_PERF 85836
computeA7: computeA7: PERFORMANCE_FAILURE: 69105 MB/s;
computeA7: computeA7: Expected_DAXPY=85836  OMP_NUM_THREADS=24 perf (DAXPY BW)=
69105
computeA7: computeA7: PERFORMANCE_FAILURE: 69105 MB/s;
```

5.3.8 The run_dgemm check

The `run_dgemm` check has the following characteristics:

Intrusive	This check is *intrusive*.
Supported processing method	Plain.
Purpose	This tool checks the CPU performance on a compute node.

Further details about the underlying `dgemm` check can be found in Appendix A, “Commonly used tools” on page 75.

Example 5-19 uses default values and lets the `dgemm` test determine the expected performance rate. All nodes fulfill the expected performance. This example uses additional arguments to overwrite the default values (the same syntax also works for the “The run_daxpy check” on page 51).

Example 5-19 The run_dgemm check with optional arguments

```
# ./hcrun -n redbook -t run_dgemm -v -u poeuser -d /root/systemcheck/bin
computeA4:   Cores=16 perf (GFlop)=  369
computeA2:   Cores=16 perf (GFlop)=  368
computeA6:   Cores=16 perf (GFlop)=  368
computeA8:   Cores=16 perf (GFlop)=  368
computeA5:   Cores=16 perf (GFlop)=  368
computeA3:   Cores=16 perf (GFlop)=  368
computeA1:   Cores=16 perf (GFlop)=  368
computeA7:   Cores=16 perf (GFlop)=  367
PERFORMANCE_SUCCESS: EXPECTED_PERF 311 : computeA4 369 GF
PERFORMANCE_SUCCESS: EXPECTED_PERF 311 : computeA2 368 GF
PERFORMANCE_SUCCESS: EXPECTED_PERF 311 : computeA6 368 GF
PERFORMANCE_SUCCESS: EXPECTED_PERF 311 : computeA8 368 GF
PERFORMANCE_SUCCESS: EXPECTED_PERF 311 : computeA5 368 GF
PERFORMANCE_SUCCESS: EXPECTED_PERF 311 : computeA3 368 GF
PERFORMANCE_SUCCESS: EXPECTED_PERF 311 : computeA1: 368 GF
PERFORMANCE_SUCCESS: EXPECTED_PERF 311 : computeA7 367 GF
```

5.4 Ethernet network: Port status, speed, bandwidth, and port errors

In this section, we show which tools can be used to determine the status of the network infrastructure. Because tests for the switch infrastructure are different from vendor to vendor, we focus on the compute nodes. As of the time of writing this book, the CHC toolkit does not have any tools to check Ethernet, so this section does not use the CHC toolkit to perform the checks. However, the following examples could easily be added to the CHC framework as new tools.

5.4.1 Ethernet firmware and drivers

Firmware and drivers versions of an Ethernet adapter can be checked with the **ethtool** Linux command. Using the **-i** parameter provides driver information for the specified interface:

```
# ethtool -i eth0
driver: bnx2
version: 2.2.1
firmware-version: bc 5.2.3 NCSI 2.0.10
bus-info: 0000:0b:00.0
```

Example 5-20 demonstrates how the **ethtool** command can be used in a clustered environment to get information about all of the Ethernet interfaces in the compute nodes. This helps in determining differences and, for example, might show back-level firmware or drivers. The example is limited because it does not take into account that different types of adapters might be in the compute nodes.

Example 5-20 Running ethtool on a group of nodes

```
# xdsh redbook 'for i in $(ls -1d /sys/class/net/eth[0-255]); do DEVICE=$(basename
$i);echo $DEVICE;ethtool -i $DEVICE; done'|xcoll
====================================
computeB3,computeB2,computeB1,computeB4
====================================
eth0
driver: bnx2
version: 2.2.1
firmware-version: bc 5.0.6 NCSI 2.0.3
bus-info: 0000:0b:00.0
eth1
driver: bnx2
version: 2.2.1
firmware-version: bc 5.0.6 NCSI 2.0.3
bus-info: 0000:0b:00.1

====================================
computeB5,computeB6
====================================
eth0
driver: bnx2
version: 2.2.1
firmware-version: bc 5.2.2 NCSI 2.0.6
bus-info: 0000:0b:00.0
```

```
eth1
driver: bnx2
version: 2.2.1
firmware-version: bc 5.2.2 NCSI 2.0.6
bus-info: 0000:0b:00.1
```

The **lspci|grep Ethernet** command (see Example 5-21) can be used to get information about the different types and models of Ethernet adapters in that compute node.

Example 5-21 Command to get information about the type and model of Ethernet adapters

```
# xdsh redbook 'lspci|grep Ethernet'|xcoll
====================================
redbook
====================================
0b:00.0 Ethernet controller: Broadcom Corporation NetXtreme II BCM5709 Gigabit
Ethernet (rev 20)
0b:00.1 Ethernet controller: Broadcom Corporation NetXtreme II BCM5709 Gigabit
Ethernet (rev 20)
```

5.4.2 Ethernet port state

The link state of a specific Ethernet interface is best checked with the **ethtool** command. The command shows the actual link speed, the duplex settings, and more, as shown in Example 5-22.

Example 5-22 The ethtool output with link up

```
# ethtool eth0
Settings for eth0:
   Supported ports: [ TP ]
   Supported link modes:   10baseT/Half 10baseT/Full
                           100baseT/Half 100baseT/Full
                           1000baseT/Full
   Supports auto-negotiation: Yes
   Advertised link modes:  10baseT/Half 10baseT/Full
                           100baseT/Half 100baseT/Full
                           1000baseT/Full
   Advertised pause frame use: No
   Advertised auto-negotiation: Yes
   Speed: 1000Mb/s
   Duplex: Full
   Port: Twisted Pair
   PHYAD: 1
   Transceiver: internal
   Auto-negotiation: on
   MDI-X: Unknown
   Supports Wake-on: g
   Wake-on: g
   Link detected: yes
```

In contrast to Example 5-22 on page 54, Example 5-23 shows an adapter with no link.

Example 5-23 The ethtool output with link down

```
# ethtool eth1
Settings for eth1:
   Supported ports: [ TP ]
   Supported link modes:   10baseT/Half 10baseT/Full
                           100baseT/Half 100baseT/Full
                           1000baseT/Full
   Supports auto-negotiation: Yes
   Advertised link modes:  10baseT/Half 10baseT/Full
                           100baseT/Half 100baseT/Full
                           1000baseT/Full
   Advertised pause frame use: No
   Advertised auto-negotiation: Yes
   Speed: Unknown!
   Duplex: Half
   Port: Twisted Pair
   PHYAD: 1
   Transceiver: internal
   Auto-negotiation: on
   MDI-X: Unknown
   Supports Wake-on: g
   Wake-on: g
   Link detected: no
```

Again, the **ethtool** command can be run on a couple of nodes to determine if all of the network connections at least have an up link at the wanted speed (Example 5-24). The command might also show if the expected network devices are all available. Additionally, if the remote shell is not working to a particular node, it shows that there might be an issue with the management network.

Example 5-24 Running ethtool through the cluster

```
# xdsh redbook 'ethtool eth1|grep -E "(Speed|Duplex|Link detected)"'|xcoll
====================================
redbook
====================================
   Speed: Unknown!
   Duplex: Half
   Link detected: no
```

5.4.3 Network settings

Network-specific settings can be modified in several ways, depending on the layer that you are looking at. There are interface-specific settings, kernel module settings, and the settings for Transmission Control Protocol (TCP), Internet Protocol (IP), User Datagram Protocol (UDP), and so on. Again, it is vital that the settings are consistent throughout the cluster.

This does not mean that they must be the same on every node, because there might be differences in the hardware. However, in most cases, they should be the same on the same hardware and software levels. Especially in heterogeneous environments, it is possible that necessary changes to default settings are not propagated on all affected nodes.

A tool, such as the configuration check tool described in “Ethernet network: Port status, speed, bandwidth, and port errors” on page 53, might be helpful to ensure network-specific settings. Table 5-2 shows some of the tools and options to change parameters in the different layers of the networking stack.

Table 5-2 Network settings

Layer	Tool	File
Network adapter	**ethtool**	/etc/sysconfig/network/ifcfg-<interface>
Kernel module	**modprobe**, **insmod**	/etc/modprobe.d/*.conf
Protocol	**sysctl**	/etc/sysctl.conf /proc/sys/net/* (temporary changes)

5.4.4 Bonding

Bonding or ether channeling is the mechanism that is used to tie network interfaces together so that they form a new device. The purpose is either to provide a higher level of redundancy, or to increase the network throughput. For example, if there are two network adapters in the compute node and each is connected to a different switch in the network, an active-backup configuration can be created. Then the network connection remains active even when an adapter, a cable, or a switch fails.

Note: The setup and configuration of bonding devices is beyond the scope of this document.

Example 5-25 shows the status of a bonding interface configured as active-backup with two network adapters. Each configured bond device has an entry in the `/proc` file system in the `net/bonding` directory.

Example 5-25 Status of a bonding interface configured as active backup

```
# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.6.0 (September 26, 2009)

Bonding Mode: fault-tolerance (active-backup)
Primary Slave: None
Currently Active Slave: eth0
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: eth0
MII Status: up
Speed: 1000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: e4:1f:13:32:4c:d4
Slave queue ID: 0

Slave Interface: eth1
MII Status: down
Speed: Unknown
```

```
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: e4:1f:13:32:4c:d6
Slave queue ID: 0
```

In Example 5-25, the secondary interface is down for some reason. But eth0 is still up and running, and the IP address configured on the bond device is still available.

In the cluster context, a distributed command can be run to show all bonded interfaces where a link might be missing. This might not affect the cluster in an active-backup configuration, but can have performance impacts when an active-active device loses one interface.

5.5 InfiniBand: Port status, speed, bandwidth, port errors, and subnet manager

This section covers the tests which can be performed regarding the InfiniBand (IB) environment of the cluster. These checks run on the compute nodes and on the IB switches.

5.5.1 The hca_basic check

The `hca_basic` check has the following characteristics:

Intrusive	The check is non-intrusive.
Supported processing methods	Plain, xcoll, and compare.
Purpose	The `hca_basic` check lists information about the IB adapters in the compute node as shown in Example 5-26. These include the vendor, model, and firmware version. Port information and connection speed is also displayed. Running this check on a node group gives a quick overview about the existing host channel adapter (HCA) in the cluster, and if that is still the wanted setup.

In Example 5-26, the link rate of computeA7 dropped to 4X Quad Data Rate (QDR). Different results could be a configuration or hardware error. For example, unsteady link rates could point to broken InfiniBand cables.

Example 5-26 The hca_basic check run on a node group with the xcoll processing method

```
# ./hcrun -n redbook -m xcoll -t hca_basic
====================================
computeA7
====================================
OFED level:     MLNX_OFED_LINUX-2.0-2.0.5 (OFED-2.0-2.0.5)
mlx4_0:
        Adapter type:   Mellanox Technologies MT27500 Family [ConnectX-3]
        Adapter fw:     2.11.1260
        HW revision:    1
        PCI speed:      Gen3
        Port 1:
                State:          PORT_ACTIVE
                Active_mtu:     4096
                Rate:           40 Gb/sec (4X QDR)
```

```
                Link layer:     InfiniBand
                Subnet prefix:  fe80:0000:0000:0000
        Port 2:
                State:          PORT_DOWN

==================================
computeA1,computeA2,computeA3,computeA4,computeA5,computeA6
==================================
OFED level:     MLNX_OFED_LINUX-2.0-2.0.5 (OFED-2.0-2.0.5)
mlx4_0:
        Adapter type:   Mellanox Technologies MT27500 Family [ConnectX-3]
        Adapter fw:     2.11.1260
        HW revision:    1
        PCI speed:      Gen3
        Port 1:
                State:          PORT_ACTIVE
                Active_mtu:     4096
                Rate:           40 Gb/sec (4X FDR10)
                Link layer:     InfiniBand
                Subnet prefix:  fe80:0000:0000:0000
        Port 2:
                State:
PORT_DOWN
```

5.5.2 The ipoib check

The `ipoib` check has the following characteristics:

Intrusive	The check is non-intrusive.
Supported processing methods	Plain, xcoll, compare.
Purpose	The `ipoib` check shows the state, maximum transmission unit (MTU), and mode of the configured IP over InfiniBand (IPoIB) devices on the compute node (Example 5-27).

Example 5-27 The ipoib with xcoll processing method

```
# ./hcrun -n redbook -m xcoll -t ipoib
==================================
redbook
==================================
IPoIB: ib0
State: UP
MTU: 2044
Mode: datagram
IPoIB: ib1
State: UP
MTU: 4092
Mode: datagram
Recv_queue_size:        512
Send_queue_size:        512
```

5.5.3 The switch_module check

All `switch_` tools are non-intrusive, support only plain mode, and have the following syntax:

```
Usage: # hcrun [-n IB switches] [-v] -t switch_....
```

To see the output while running the tools, use the **-v** option. If no InfiniBand switches or InfiniBand switch node groups are specified with the **-n** parameter, the switches defined by the **ib_switches** setting in `/etc/opt/ibmchc/chc.conf` are used, as shown in Example 5-28. The command lists the different modules that are installed in the given switch chassis. The result is written to a log file and not to the console.

Example 5-28 Switch modules

```
# ./hcrun -n ibswA1 -t switch_module
Results will be logged to:  /var/opt/ibmchc/log/swmodule.2013-12-11-17:23:17

# cat /var/opt/ibmchc/log/swmodule.2013-12-11-17:23:17
================================================================================
ibswA1: Module              Type                Present             Power
================================================================================
ibswA1: MGMT2               SX6000              1                   N/A
ibswA1: MGMT1               IS5600MDC           0                   N/A
ibswA1: S01                 SX6002              1                   1
ibswA1: S02                 SX6002              1                   1
ibswA1: S03                 SX6002              1                   1
ibswA1: S04                 SX6002              1                   1
ibswA1: S05                 SX6002              1                   1
ibswA1: S06                 SX6002              1                   1
ibswA1: L01                 SX_LEAF             0                   0
ibswA1: L02                 SX_LEAF             0                   0
ibswA1: L03                 SX_LEAF             0                   0
ibswA1: L04                 SX_LEAF             0                   0
ibswA1: L05                 SX6001              1                   1
ibswA1: L06                 SX6001              1                   1
ibswA1: L07                 SX6001              1                   1
ibswA1: L08                 SX6001              1                   1
ibswA1: L09                 SX6001              1                   1
ibswA1: L10                 SX6001              1                   1
ibswA1: L11                 SX6001              1                   1
ibswA1: L12                 SX6001              1                   1
ibswA1: FAN1                IS5X00_FAN          1                   N/A
ibswA1: FAN2                IS5X00_FAN          1                   N/A
ibswA1: FAN3                IS5X00_FAN          1                   N/A
ibswA1: FAN4                IS5X00_FAN          1                   N/A
ibswA1: PS1                 PS1648              1                   N/A
ibswA1: PS2                 PS1648              1                   N/A
ibswA1: PS3                 PS1648              1                   N/A
ibswA1: PS4                 PS1648              1                   N/A
ibswA1: CPU                 CPU                 1                   N/A
```

5.5.4 The switch_ntp check

The `switch_ntp` check verifies that Network Time Protocol (NTP) is configured and active on the InfiniBand switches, as shown in Example 5-29. Although not really necessary for the operation of the network, it eases problem determination if all of the time stamps in the logs are in sync throughout the cluster.

Example 5-29 NTP active but not synchronized

```
# ./hcrun -n ibswA1 -t switch_ntp
====================================
ibswA1
====================================
Node count = 1
====================================
NTP is enabled.
Clock is unsynchronized.
No NTP peers or servers configured.
```

5.5.5 The switch_inv check

This tool lists the installed modules in a switch, together with their part and serial number, as shown in Example 5-30. The output is written to a log file and not to the console.

Example 5-30 Switch inventory

```
# ./hcrun -n ibswA1 -t switch_inv
Results will be logged to:  /var/opt/ibmchc/log/swinv.2013-12-11-17:23:17

# cat /var/opt/ibmchc/log/swinv.2013-12-11-17:23:17
=============================================================================================
ibswA1: Module               Type                Part number          Serial Number
=============================================================================================
ibswA1: CHASSIS              SX6512              00W0012              00W0012YK5020C00001
ibswA1: MGMT2                SX6000              90Y3804              90Y3791YK502000004S
ibswA1: S01                  SX6002              90Y3846              90Y3789YK50200000HN
ibswA1: S02                  SX6002              90Y3846              90Y3789YK50200000HX
ibswA1: S03                  SX6002              90Y3846              90Y3789YK50200000HS
ibswA1: S04                  SX6002              90Y3846              90Y3789YK50200000KK
ibswA1: S05                  SX6002              90Y3846              90Y3789YK50200000F9
ibswA1: S06                  SX6002              90Y3846              90Y3789YK50200000KB
ibswA1: L05                  SX6001              MSX6001FR            MT1233X00822
ibswA1: L06                  SX6001              MSX6001FR            MT1245X07530
ibswA1: L07                  SX6001              MSX6001FR            MT1233X00824
ibswA1: L08                  SX6001              MSX6001FR            MT1232X03564
ibswA1: L09                  SX6001              MSX6001TR            MT1226X00086
ibswA1: L10                  SX6001              MSX6001TR            MT1226X00085
ibswA1: L11                  SX6001              90Y3806              90Y3807YK5020000140
ibswA1: L12                  SX6001              90Y3806              90Y3807YK5020000141
ibswA1: FAN1                 IS5X00_FAN          81Y1500              81Y1500YK5020C00007
ibswA1: FAN2                 IS5X00_FAN          81Y1500              81Y1500YK5020C00008
ibswA1: FAN3                 IS5X00_FAN          81Y1500              81Y1500YK5020C00009
ibswA1: FAN4                 IS5X00_FAN          81Y1500              81Y1500YK5020C00010
ibswA1: PS1                  PS1648              900-1648-00          00322
ibswA1: PS2                  PS1648              900-1648-00          00323
ibswA1: PS3                  PS1648              900-1648-00          00324
ibswA1: PS4                  PS1648              900-1648-00          00325
ibswA1: CPU                  CPU                 SA000203-B           MT1229X01207
```

5.5.6 The switch_health check

The `switch_health` check reports the health of the switches given in the node list, as shown in Example 5-31. The output is written to a log file and not to the console. This example shows voltage problems of some *leafs*.

Example 5-31 Health report

```
# ./hcrun -n ibswA1 -t switch_health
Results will be logged to:  /var/opt/ibmchc/log/swhealth.2013-12-11-17:43:23

# cat /var/opt/ibmchc/log/swhealth.2013-12-11-17:43:23
ibswA1: HEALTH DAEMON REPORT
ibswA1: ========================
ibswA1: ERR 2013/04/15 13:55:52 : Leaf number 12 voltage is out of range
ibswA1: NTC 2013/04/15 13:56:52 : Leaf number 12 voltage is in range
ibswA1: ERR 2013/05/21 11:28:52 : Leaf number 11 voltage is out of range
ibswA1: NTC 2013/05/21 11:29:51 : Leaf number 11 voltage is in range
```

5.5.7 The switch_clk check

By using the time of the management station, and given a clock skew that can be set with the **`HC_CLKSKEW_ALLOWED`** environment variable, the `switch_clk` tool determines if the time is consistent throughout the cluster. In Example 5-32, a mismatch is detected.

Requirement: To see output, the verbosity option (**`-v`**) is needed.

Example 5-32 The switch_clk check reports an error

```
# ./hcrun -n ibswA1+1 -v -t switch_clk
Local timestamp 2013/12/11 17:56:28 EST; target skew <= 10 seconds

Switches in group ibswA1+1 out of skew ( >10 seconds )
 Total = 2 switch(es)
 Default clock skew was used; <= 10 seconds
 Depending on the number of switches, a larger value may be required
 - ibswA2 time 2013/12/11 22:51:22 UTC (-306 seconds)
 - ibswA1 time 2013/07/14 22:13:25 UTC (-12962583 seconds)
The health check tool switch_clk                                [  FAILED  ]
```

5.5.8 The switch_code check

The `switch_code` check evaluates whether the switch code is consistent, as shown in Example 5-33.

Example 5-33 Switch code check

```
# ./hcrun -n ibswA1+1 -t switch_code
Results will be logged to:  /var/opt/ibmchc/log/swcode.2013-12-11-18:01:47

# cat /var/opt/ibmchc/log/swcode.2013-12-11-18:01:47
====================================
ibsw
```

```
====================================
Node count = 2
====================================
Product name:      SX_PPC_M460EX
Product release:   SX_3.2.0330-100

====================================
ibswA2
====================================
Node count = 1
====================================
SX module           Version
SO1                 9.1.6516
SO2                 9.1.6516
SO3                 9.1.6516
SO4                 9.1.6516
SO5                 9.1.6516
SO6                 9.1.6516
LO8                 9.1.6516
LO9                 9.1.6516
L10                 9.1.6516
L11                 9.1.6516
L12                 9.1.6516

====================================
ibswA1
====================================
Node count = 1
====================================
SX module           Version
SO1                 9.1.6516
SO2                 9.1.6516
SO3                 9.1.6516
SO4                 9.1.6516
SO5                 9.1.6516
SO6                 9.1.6516
LO5                 9.1.6516
LO6                 9.1.6516
LO7                 9.1.6516
LO8                 9.1.6516
LO9                 9.1.6516
L10                 9.1.6516
L11                 9.1.6516
L12                 9.1.6516
```

5.5.9 The run_ppping check

The `run_ppping` check has the following characteristics:

Intrusive	The check is *intrusive*.
Supported processing method	Plain.
Purpose	The run_ppping check determines if the configured IPoIB devices are configured and reply to local ping requests. All given nodes ping each other.

In Example 5-34, three nodes do not respond to the ping requests. An additional output file is created, and is located under `/var/opt/ibmchc/data/ppping.ib0.out`.

Example 5-34 The run_ppping failed

```
# ./hcrun -n redbook -t run_ppping
Error: ib interface ib0 is not active on node computaA1
Error: ib interface ib0 is not active on node computeA3
Error: ib interface ib0 is not active on node computeA7
Error: could not ping over all ib0 interfaces
Info: ppping output can be found in /var/opt/ibmchc/data/ppping.ib0.out
The health check tool run_ppping                                  [  FAILED  ]
```

Example 5-35 shows the output file.

Example 5-35 The run_ppping output file

```
computaA2: computaA1-ib0: noping
computaA2: computeA3-ib0: noping
computaA2: computeA7-ib0: noping
computaA3: computaA1-ib0: noping
computaA3: computeA3-ib0: noping
computaA3: computeA7-ib0: noping
...
```

The ib0 interfaces of computeA1, computeA3, and computeA7 need additional research.

5.5.10 The run_jlink check

The `run_jlink` check has the following characteristics:

Intrusive	The check is *intrusive*.
Supported processing method	Plain.
Purpose	The `jlink` tool is used to measure the bandwidth between compute nodes, and to discover bad or low-performing links. The tool has a wide variety of options, which are described in Appendix A, “Commonly used tools” on page 75.

When `jlink` is started through **`hcrun`**, it is necessary that the compute nodes have a working PE installed. Because PE jobs should not be run as root, a dedicated user must be available to run this check. This user can either be specified in `/etc/chc/chc.conf` (see “Configuration” on page 38 for details), or on the command line when running this check.

Example 5-36 shows a run on an eight-node cluster using the `jlink` default settings. Because there is no user defined in the global configuration file, the `hcrunuser` user is provided on the command line. These users exist on all nodes.

Example 5-36 The jlink tool run on an eight-node cluster

```
# ./hcrun -n redbook -t run_jlink -u hcrunuser
User requested NUM_RUNS=30
User requested no thresholding THRESH_FLAG=
User requested REPS_STEADY=300
Based on highest ibstat reported 'Rate', selected threshold flag = -L 4250
poe /opt/ibmhpc/pecurrent/systemcheck/bin/jlink -s 1 -n 30 -r 300 -w 1 -u 1000 -U 1000000 -L 4250
   0:Loop: 0 =   4 secs  (   4 total)  Avg_bw=4292 MB/s  Agg_bw=240358 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
```

```
   0:Loop: 1 =   4 secs  (   8 total)  Avg_bw=4299 MB/s  Agg_bw=240738 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 2 =   4 secs  (  12 total)  Avg_bw=4300 MB/s  Agg_bw=240799 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 3 =   4 secs  (  16 total)  Avg_bw=4300 MB/s  Agg_bw=240817 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 4 =   4 secs  (  20 total)  Avg_bw=4298 MB/s  Agg_bw=240685 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 5 =   4 secs  (  24 total)  Avg_bw=4303 MB/s  Agg_bw=240986 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 6 =   4 secs  (  28 total)  Avg_bw=4302 MB/s  Agg_bw=240887 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 7 =   4 secs  (  32 total)  Avg_bw=4303 MB/s  Agg_bw=240953 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 8 =   4 secs  (  35 total)  Avg_bw=4300 MB/s  Agg_bw=240804 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 9 =   4 secs  (  39 total)  Avg_bw=4302 MB/s  Agg_bw=240919 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 10 =   4 secs  (  43 total)  Avg_bw=4301 MB/s  Agg_bw=240833 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 11 =   4 secs  (  47 total)  Avg_bw=4302 MB/s  Agg_bw=240905 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 12 =   4 secs  (  51 total)  Avg_bw=4301 MB/s  Agg_bw=240829 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 13 =   4 secs  (  55 total)  Avg_bw=4300 MB/s  Agg_bw=240810 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 14 =   4 secs  (  59 total)  Avg_bw=4300 MB/s  Agg_bw=240787 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 15 =   4 secs  (  63 total)  Avg_bw=4300 MB/s  Agg_bw=240795 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 16 =   4 secs  (  67 total)  Avg_bw=4300 MB/s  Agg_bw=240778 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 17 =   4 secs  (  71 total)  Avg_bw=4300 MB/s  Agg_bw=240799 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 18 =   4 secs  (  75 total)  Avg_bw=4300 MB/s  Agg_bw=240803 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 19 =   4 secs  (  79 total)  Avg_bw=4303 MB/s  Agg_bw=240954 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 20 =   4 secs  (  83 total)  Avg_bw=4301 MB/s  Agg_bw=240857 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 21 =   4 secs  (  87 total)  Avg_bw=4299 MB/s  Agg_bw=240764 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 22 =   4 secs  (  91 total)  Avg_bw=4300 MB/s  Agg_bw=240825 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 23 =   4 secs  (  95 total)  Avg_bw=4299 MB/s  Agg_bw=240771 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 24 =   4 secs  (  99 total)  Avg_bw=4301 MB/s  Agg_bw=240844 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 25 =   4 secs  ( 102 total)  Avg_bw=4301 MB/s  Agg_bw=240868 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 26 =   4 secs  ( 106 total)  Avg_bw=4301 MB/s  Agg_bw=240879 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 27 =   4 secs  ( 110 total)  Avg_bw=4302 MB/s  Agg_bw=240913 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 28 =   4 secs  ( 114 total)  Avg_bw=4302 MB/s  Agg_bw=240885 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
   0:Loop: 29 =   4 secs  ( 118 total)  Avg_bw=4301 MB/s  Agg_bw=240838 MB/s Loop_bw=  38 MB/s  Exchanges=    56
Active_nodes=   8
Checking file: out.all.8.1.1011.054239

Tested 56 IB paths and found 0 below threshold (0.00 %).
Tested 56 IB paths out of 56 (untested: ) (tested: 100.00 %).
The health check tool run_jlink  [ PASS ]
```

5.5.11 The run_ibtools check

The `run_ibtools` check is not a single tool, it is a group of multiple tools. The syntax for groups is a little bit different:

```
# hcrun -n <noderange> <group name>
```

Intrusive	The check is *intrusive*.
Supported processing method	Plain.
Purpose	This tool measures the bandwidth and reads bit error rates, then correlates both and shows suspicious links.

Before run_ibtools is able to work, you have to run the ibtools_install tool once. It installs the ibtools on the subnet manager, and creates the /var/opt/ibtools/data output directory on the management node. The ibtools are in the RPM, and deployed on the management node. The ibtools_install tool takes these scripts and copies them to the subnet manager.

The run_ibtools group runs the following tools in order:

1. clearerrors
   Clears the data and error counters on the subnet manager. This is needed before an ibqber can be run. The bit error rate (BER) is determined relative to the last time the error counters had been cleared on the subnet manager.
2. lastclear
   This tool is used to display the last time the error and data counters were cleared on the subnet manager.
3. run_jlink
   See “The run_jlink check” on page 63.
4. ibqber
   This runs an ibquery and collects BER data on a subnet manager. This should be run after a thorough exercise of the fabric through run_jlink or some other exerciser to accumulate error counts on fabric links. This might or might not be the case, so meaningful output is not guaranteed. If the network is clean, there will not be any errors.
5. berlinks
   This takes a file that has a BER links calculation that results from running ibqber on the subnet manager. The default is the last *.BER.1e-14 file for a particular subnet manager in /var/opt/ibtools/data on the management node. It shows links with BER data. The berlinks output files are input to jlinksuspects.
6. jlinkpairs
   This tool takes a jlink output file produced by running jlink, or uses the default jlink.out file in /var/opt/ibmchc/data. It shows slow links. The jlinkpairs output file is input to jlinksuspects.
7. jlinksuspects
   This is the final result of run_ibtools (Example 5-37). It takes a jlinkpairs output file and a berlinks output file to produce an output listing showing the links with a higher-than-expected BER that exist in the path between a pair of nodes that have lower-than-expected bandwidth.
8. ibtoolslog_clean
   This tool archives the accumulated data files in /var/opt/ibtools/data on the management node and the subnet manager.

Example 5-37 The run_ibtools check with 2 nodes

```
# ./hcrun -n computeA1,computeA2 run_ibtools
Info: clearerrors output can be found in /var/opt/ibtools/data/sm01.clear.log
The health check tool clearerrors                              [ PASS ]
Info: lastclear output can be found in /var/opt/ibtools/data/sm01.log.counterclears
The health check tool lastclear                                [ PASS ]
Info: jlink output can be found in /var/opt/ibmchc/data/jlink.out
The health check tool run_jlink                                [ PASS ]
Info: ibqber output can be found in /var/opt/ibtools/data/ibq.20131218_093743
Info: ibqber output can be found in /var/opt/ibtools/data/ibq.20131218_093743.BER.1e-14
Info: ibqber output can be found in /var/opt/ibtools/data/ibq.20131218_093743.BER.5e-14
The health check tool ibqber                                 [ PASS ]
Info: berlinks output can be found in /var/opt/ibtools/data/sm01.all.berlinks.out
The health check tool berlinks                               [ PASS ]
Info: jlinkpairs output can be found in /var/opt/ibtools/data/jlinkpairs.out
The health check tool jlinkpairs                               [ PASS ]
sm01:
Pairs: computeA1<->computeA2 computeA2<->computeA1
```

```
Links: c902ibsw01/L12/U1:13<->c902f04x25 HCA-1:1 computeA1 HCA-1:1<->c902ibsw01/L10/U1:12 computeA2
HCA-1:1<->c902ibsw01/L10/U1:9 c902ibsw01/L07/U1:13<->c902f02x55 HCA-1:1 c902f04x60-ib0
HCA-1:1<->c902ibsw01/L11/U1:5
------------------------------------------------------------
- Check if any node pair has a suspect node or link
------------------------------------------------------------
Info: berlinks output can be found in /var/opt/ibtools/data/sm01.node.berlinks.out
computeA1 in node pairs
computeA2 in node pairs
------------------------------------------------------------
- Check if any node pair has a suspect switch-to-switch link
------------------------------------------------------------
- Check computeA1 HCA-1 to computeA2 HCA-1
computeA1 HCA-1 (29) port 1 computeA2 HCA-1 (26) port 1
- Check computeA1 HCA-1 to computeA2 HCA-1
computeA1 HCA-1 (29) port 1 MF0;c902ibsw01:SXX512/L10/U1 (61) port 12 computeA2 HCA-1 (26) port 1
- Check computeA1 HCA-1 to computeA2 HCA-1
computeA1 HCA-1 (29) port 1 MF0;c902ibsw01:SXX512/L10/U1 (61) port 9 computeA2 HCA-1 (26) port 1
- Check computeA1 HCA-1 to computeA2 HCA-1
computeA1 HCA-1 (29) port 1 computeA2 HCA-1 (26) port 1
- Check computeA1 HCA-1 to computeA2 HCA-1
computeA1 HCA-1 (29) port 1 computeA2 HCA-1 (26) port 1
- Check computeA2 HCA-1 to computeA1 HCA-1
computeA2 HCA-1 (26) port 1 computeA1 HCA-1 (29) port 1
- Check computeA2 HCA-1 to computeA1 HCA-1
computeA2 HCA-1 (26) port 1 MF0;c902ibsw01:SXX512/L10/U1 (61) port 12 computeA1 HCA-1 (29) port 1
- Check computeA2 HCA-1 to computeA1 HCA-1
computeA2 HCA-1 (26) port 1 MF0;c902ibsw01:SXX512/L10/U1 (61) port 9 computeA1 HCA-1 (29) port 1
- Check computeA2 HCA-1 to computeA1 HCA-1
computeA2 HCA-1 (26) port 1 computeA1 HCA-1 (29) port 1
- Check computeA2 HCA-1 to computeA1 HCA-1
computeA2 HCA-1 (26) port 1 computeA1 HCA-1 (29) port 1
The health check tool jlinksuspects                              [ PASS ]
Local: /var/opt/ibtools/data
 >1 day old
  5
 Info: Number of log files below quota of 10
here?
sm01: /var/opt/ibtools/data
 >1 day old
  4
 Info: Number of log files below quota of 10
The health check tool ibtoolslog_clean                           [ PASS ]
```

5.6 File system: Accessibility, usage, and read/write performance

In most cases, the cluster nodes share a common file system across all of the nodes. Possible solutions are the IBM General Parallel File System (GPFS), the IBM Network File System (NFS), Lustre, Glusterfs, or similar cluster file systems. In this document, we focus on IBM GPFS and NFS.

In this section, we look at the basic instruments to check for the availability of required file systems. Tools, such as **nsdperf** or **gpfsperf**, are addressed in more detail in “GPFS related tools” on page 83.

5.6.1 The fs_usage check

The `fs_usage` check has the following characteristics:

Intrusive	The check is non-intrusive.
Supported processing methods	Plain, xcoll, and compare.
Purpose:	The fs_usage check assesses the used file system space of the provided paths. Per default, it checks `/tmp` and `/var/tmp` (Example 5-38). To check other paths, edit the `/opt/ibmchc/tools/fs_usage.conf` tool configuration file, or append your own arguments behind the toolname, as shown in Example 5-39.

Example 5-38 The fs_usage check with default arguments

```
./hcrun -n computeA1 -t fs_usage
computeA1: /tmp: 3%
computeA1: /var/tmp: 0%
```

Example 5-39 The fs_usage check with custom arguments

```
./hcrun -n computeA1 -t fs_usage "/home /root"
computeA1: /home: 6%
computeA1: /root: 6%
```

5.6.2 NFS file system

NFS file systems are provided by an NFS server and mounted on NFS clients over the network. Current Linux distributions support NFS versions 2, 3, and 4. In many cases, an NFS server is configured for high availability to avoid a single point of failure. The setup and configuration of an NFS server is nevertheless beyond the scope of this document.

To check if an NFS-mounted file system is accessible, a simple read/write check can be performed, as shown in Example 5-40.

Example 5-40 Simple read/write check

```
$ dd if=/dev/urandom of=dummy1.dat bs=1M count=100
100+0 records in
100+0 records out
104857600 bytes (105 MB) copied, 11.752 s, 8.9 MB/s
$ dd if=dummy1.dat of=/dev/null
204800+0 records in
204800+0 records out
104857600 bytes (105 MB) copied, 0.112903 s, 929 MB/s
```

The following list includes some issues that might occur with an NFS resource:

- Inconsistent mounting on the compute nodes
- Mount options
- Network issues

NFS mounts

Although NFS mounts appear to be obvious, you should verify that they are actually mounted on all of the necessary compute nodes. Depending on how the NFS file system is mounted, the verification can include whether the respective files or scripts are in sync. For example, if `/etc/fstab`, or a startup script that mounts a specific resource, is available on all nodes. Even if this is the case, there might be other reasons why a mount is no longer valid.

A simple distributed shell command checks for inconsistencies, as shown in Example 5-41.

Example 5-41 List NFS-mounted file systems

```
# xdsh redbook "mount -t nfs"|xcoll
====================================
computeA2,computeA8,computeA6,computeA4,computeA7,computeA1,computeA3
====================================
ugpfs01-ug:/gpfs/cnfs01/data/u on /u type nfs (rw,vers=3,addr=60.0.2.191)
ugpfs01-ug:/gpfs/cnfs01/data/admin on /ugadmin type nfs (rw,vers=3,addr=60.0.2.191)

====================================
computeA5
====================================
ugpfs01-ug:/gpfs/cnfs01/data/u on /u type nfs (rw,vers=3,addr=60.0.2.191)
```

In Example 5-41, the file system `/u` is not mounted on computeA5. This might lead to inconsistent behavior in the cluster, but does not necessarily lead to an immediate failure.

The CHC toolkit has an NFS mount check too. The tool is called `nfs_mounts`. The output is similar to that shown in Example 5-42. This tool is non-intrusive, and supports all three processing methods (plain, xcoll, and compare).

Example 5-42 CHC nfs_mounts tool

```
# ./hcrun -n computeA1 -t nfs_mounts
computeA1: ugpfs03-ug:/gpfs/cnfs01/data/u
computeA1: ugpfs03-ug:/gpfs/cnfs01/data/admin
```

Important: Although a file system appears to be mounted, it is not necessarily accessible.

If a file system is not mounted on all of the nodes, check if the NFS server still exports the file system, or if that has been changed with the following command, which shows all of the file systems that the NFS server exports:

```
# showmount -e <NFS Server>
```

Mount options

NFS file systems are often mounted with specific parameters: for example **`rsize`**, **`wsize`**, **`proto`**, **`soft/hard`**, and so on. Usually, they should be the same on all nodes, so differences might arise when NFS mounts are performed manually (for example, `/etc/fstab` is not in sync, or similar situations).

Network issues

Most problems with the NFS file system have their origin in the underlying network. In case there are issues, such as poor performance or hangs when accessing the file system, and all the settings appear to be fine, a closer look in the network needs to be performed.

To determine network issues, consider the following questions:

- ☐ Are the local network interfaces up and configured correctly?
- ☐ Are the NFS server network interfaces up and configured correctly?
- ☐ Do they have a link?
- ☐ Is the NFS server reachable (**ping**)?
- ☐ Is there a route to the NFS server (**traceroute**, **tracepath**)?

Failure in one of these basic checks could then lead, for example, to cable problems, blocked ports on Ethernet switches and so on. In other words, a careful end-to-end analysis is required.

5.6.3 GPFS file system

This section is not about the setup or configuration of a GPFS cluster. The information in this section is presented with the assumption that this has already been done, and that there are GPFS file systems available for the cluster. Similar to “NFS file system” on page 67, we look at a few basic checks to see if the GPFS file system is available and usable to the cluster. More tools for the measurement of the performance of a GPFS file system can be found in “GPFS related tools” on page 83.

GPFS status

This section provides GPFS status information.

Node status

The general status of the GPFS cluster is obtained with the **mmgetstate** command, as shown in Example 5-43. When run without any options, it displays the GPFS status of the node that the command is running on. Run with the **-a** option, the command displays the state of all of the nodes in the cluster.

Example 5-43 The mmgetstate command in a six-node cluster

```
# mmgetstate -a

 Node number  Node name        GPFS state
------------------------------------------
       1      computeB1       active
       2      computeB2       active
       3      computeB3       active
       4      computeB4       active
       5      computeB5       active
       6      computeB6       active
```

In Example 5-43, all of the nodes are active. If there are nodes in a `down` or `arbitrating` state, this might indicate a problem.

Disk status

A GPFS file system requires Network Shared Disks (NSDs) to be available. Although a file system can work even when one or more disks are down or unavailable, the status should be known, and therefore verified with the **mmlsdisk** command.

Example 5-44 lists all of the NSDs for the `fsB` file system.

Example 5-44 Status of NSDs in a file system

```
# mmlsdisk fsB -L
disk         driver   sector     failure holds    holds                                 storage
name         type       size       group metadata data  status        availability disk id pool         remarks
------------ -------- ------ ----------- -------- ----- ------------- ------------ ------- ------------ ---------
nsd39        nsd         512           1 Yes      Yes   ready         up                 1 system        desc
nsd41        nsd         512           1 Yes      Yes   ready         up                 2 system        desc
Number of quorum disks: 2
Read quorum value:      2
Write quorum value:     2
```

The `availability` status shows that both disks are `up` and ready, so there should not be any issue with the file system.

Note: This is an oversimplified setup for demonstration purposes only.

GPFS mounts

GPFS file systems are mounted or unmounted with the GPFS **`mmmount`** or **`mmumount`** commands. To verify on which nodes a given file system is mounted, use the **`mmlsmount`** command. Without specified options, it lists the GPFS file system currently mounted on the local node. With additional parameters (**`all -L`**), Example 5-45 lists the nodes where this or these file systems are mounted.

Example 5-45 List nodes and file system

```
# mmlsmount all -L

File system fsB is mounted on 6 nodes:
  10.3.1.39       computeB5
  10.3.1.37       computeB4
  10.3.1.33       computeB2
  10.3.1.41       computeB6
  10.3.1.35       computeB3
  10.3.1.31       computeB1
```

Example 5-45 helps to verify that a file system is mounted on all of the nodes in the cluster.

If a GPFS file system is mounted automatically on the node (at start or boot), the GPFS file system automount settings are configured for automatic mounting. These settings can be listed with the **`mmlsfs`** command. If only the current setting for the automount parameter should be queried, the **`mmlsfs <file system> -A`** command is sufficient. Example 5-46 shows that `fsB` is not mounted automatically.

Example 5-46 List automount setting for file system

```
#  mmlsfs fsB -A
flag                value                    description
------------------- ------------------------ -----------------------------------
 -A                 no                       Automatic mount option
```

The **mmlsconfig** command can be used to list the file systems that are available in the local GPFS cluster, as shown in Example 5-47.

Example 5-47 List available file systems

```
# mmlsconfig
Configuration data for cluster gpfsB.computeB5:
-----------------------------------------------
myNodeConfigNumber 6
clusterName gpfsB.computeB5
clusterId 9191008596370922699
autoload no
dmapiFileHandleSize 32
minReleaseLevel 3.5.0.7
verbsRdma enable
verbsPorts mlx4_0/1
adminMode central

File system in cluster gpfsB.computeB5:
---------------------------------------
/dev/fsB
```

If using a remote GPFS cluster, use the **mmremotecluster show all** command to display the mounted remote GPFS file systems.

The gpfs_state check

The `gpfs_state` check has the following characteristics:

Intrusive	The check is non-intrusive.
Supported processing method	Plain, xcoll, and compare.
Purpose	The `gpfs_state` check determines if the GPFS file system is mounted, and whether `verbsRDMA` is enabled.

Provide file systems as arguments, or edit the tool configuration and add default arguments.

In Example 5-48, computeA5 does not have `fsB` mounted.

Example 5-48 Checking fsB mount with gpfs_state

```
# ./hcrun -n redbook -t gpfs_state fsB
computeA2: fsB : OK
computeA1: fsB : OK
computeA3: fsB : OK
computeA4: fsB : OK
computeA5: fsB : FAIL
computeA6: fsB : OK
computeA2: verbsRDMA : OK
computeA1: verbsRDMA : OK
computeA3: verbsRDMA : OK
computeA5: verbsRDMA : OK
computeA6: verbsRDMA : OK
```

GPFS settings

As described in “GPFS mounts” on page 70, the **mmlsconfig** command is used to list the various configurations that can be adjusted in a GPFS cluster setup. Most of these settings vary depending on the special needs of the GPFS cluster.

Despite the specific setting of a parameter, it must also be noted that some settings might only apply to a subset of nodes. There might be options that only need to be set on a quorum node, and could lead to unwanted side effects when set on a non-quorum node. If settings only apply to some nodes, these appear in the **mmlsconfig** output, as shown in Example 5-49.

Example 5-49 Different settings on different nodes

```
# mmlsconfig
Configuration data for cluster gpfsB.computeB5:
-----------------------------------------------
myNodeConfigNumber 6
clusterName gpfsB.computeB5
clusterId 9191008596370922699
autoload no
[computeB5,computeB6]
autoload yes
[common]
dmapiFileHandleSize 32
minReleaseLevel 3.5.0.7
verbsRdma enable
verbsPorts mlx4_0/1
[computeB4]
unmountOnDiskFail yes
[common]
adminMode central

File system in cluster gpfsB.computeB5:
---------------------------------------
/dev/fsB
```

In Example 5-49, some settings only apply to some nodes. In large GPFS clusters, this might be a source for issues, because settings can easily be overlooked, and important settings might be missing on some nodes.

GPFS file system access

As shown in “NFS file system” on page 67, a simple check, such as the one in Example 5-40 on page 67, can be performed to determine if the mounted file system is accessible at all. As mentioned earlier, tools such as **nsdperf** or **gpfsperf** can be used to measure the performance of the file system.

Tip: Accessing a GPFS file system that suffered a failure earlier might report *Stale NFS file handle* messages. This might cause confusion when talking to support teams, especially when they have little or no knowledge about GPFS.

The run_nsdperf check

Intrusive:	The check is **intrusive**.
Supported processing method	Plain.
Purpose:	The `run_nsdperf` check is a performance tool that mimics a GPFS client/server traffic pattern on the cluster without actually using GPFS. The tool checks GPFS network performance.

Further details about the underlying **`nsdperf`** check can be found in “Example of nsdperf” on page 83.

A

Commonly used tools

This appendix provides additional information about commonly used tools in addition to those contained in the Cluster Health Check (CHC), which is described in detail in Chapter 5, “Toolkits for verifying health (individual diagnostics)” on page 37.

These tools are a useful addition to the CHC tools. This overview is certainly not meant to be a complete reference. However, the chapter describes a few important tools, and provides basic information about how to run them.

This appendix also introduces IBM High Performance Computing (HPC) Central and HPC service packs.

The following sections are described in this appendix:

- Overview of non-CHC tools
- InfiniBand-related tools
- GPFS related tools
- Network-related tools
- Disk benchmarks
- Node-specific tools
- IBM HPC Central

Overview of non-CHC tools

This section gives a short outline of the available tools and their use cases. These tools can be grouped into categories. A short overview is shown in Table A-1.

Table A-1 Overview of useful tools

Category	Tools	Description
InfiniBand	**ibdiagnet**, **jlink**	InfiniBand Fabric tests, performance tests
General Parallel File System (GPFS)	**gpfsperf**, **nsdperf**	GPFS-specific test tools delivered with the GPFS product
Network	**iperf**	Internet Protocol network tools
Disk and file system	**IOR**, **mdtest**	General file system tools
Node	**stream**, **daxpy**, **dgemm**, **OS jitter**	Tools useful to test nodes

The shown tools are by far not a complete selection. Many more tools are publicly available.

InfiniBand-related tools

There are many utilities for InfiniBand that are included with OpenFabrics Enterprise Distribution (OFED) distributions. This IBM Redbooks publication focuses on the OFED distribution provided by Mellanox, which is installed in our clusters.

The **ibdiagnet** tool is one of the most important tools to check the fabric health. This tool is used to check the routing, and is good for finding credit loops, especially those caused by mis-wires or poor topology design. The tool also reveals slow links and other common problems. The **ibdiagnet** tool performs tests within different sections, for example:

- Routing
- Topology
- Partitions
- Bit error checks
- Cable reports

In addition to the **ibdiagnet** tool, OFED provides many more tools and commands, which are outlined shortly.

The **jlink** utility includes the IBM Parallel Environment (PE). This tool is intended to isolate links with low performance, especially in large clusters, within a reasonable amount of time.

Example of ibdiagnet

Typically, **ibdiagnet** is run once to clear the counters. Then, for a period of time, such as 15-30 minutes, traffic needs to be generated by, for example, running a few applications. After this, the tool can be run again to generate a report that helps detect suspect and bad links.

During the initial setup, the tool usually is run on an idle fabric to detect link level errors for a period of time. After fixing errors for the idle fabric, the stress test should be run. Again, fix errors and repeat until **ibdiagnet** does not report any major errors. Note that it is not mandatory to have zero errors, but the number should stay at a very low level.

The following steps represent the basic workflow:

1. Run **ibdiagnet -pc** to clear the counters.
2. Create traffic by running, for example, a few benchmarks (typically for 20-30 minutes).
3. Run **ibdiagnet -r** after a period of time to collect the information again.

Example A-1 shows a run on ClusterB that reveals strongly increasing symbol errors and exceeded bit error rate (BER) values for two nodes in the fabric. Calling **ibdiagnet** as shown enables you to accomplish everything in one call. The **pm_pause_time** parameter determines how long this test is run. After starting **ibdiagnet**, traffic can be created in the fabric. The **ibdiagnet** call automatically stops after **pm_pause_time** seconds.

Note: The computeC4 node appears in the output of **ibdiagnet** as well, because ClusterB and ClusterC in reality share the same physical InfiniBand fabric.

The **-lw** and **-ls** parameters in **ibdiagnet** refer to the expected values for link width and link speed. For Fourteen Data Rate (FDR10) and Quad Data Rate (QDR), the link speed is 10. For FDR, a value of 14 is required for the **-ls** parameter.

Example A-1 The ibdiagnet command run on ClusterB

```
[root@computeB1 ~]# ibdiagnet -lw 4x -ls 10 -ber_test --ber_thresh 1000000000000
-P all=1 -pc -pm_pause_time 30
----------
Load Plugins from:
/usr/share/ibdiagnet2.1.1/plugins/
(You can specify more paths to be looked in with "IBDIAGNET_PLUGINS_PATH" env
variable)

Plugin Name                                   Result     Comment
libibdiagnet_cable_diag_plugin                Succeeded  Plugin loaded
libibdiagnet_cable_diag_plugin-2.1.1          Failed     Plugin options issue -
Option "get_cable_info" from requester "Cable Diagnostic (Plugin)" already exists
in requester "Cable Diagnostic (Plugin)"

-----------------------------------------------
Discovery
-I- Discovering ... 19 nodes (1 Switches & 18 CA-s) discovered.
-I- Fabric Discover finished successfully

-I- Discovering ... 19 nodes (1 Switches & 18 CA-s) discovered.
-I- Discovery finished successfully

-I- Duplicated GUIDs detection finished successfully

-I- Duplicated Node Description detection finished successfully

-I- Retrieving ... 19/19 nodes (1/1 Switches & 18/18 CA-s) retrieved.
-I- Switch information retrieving finished successfully
```

```
---------------------------------------------
Lids Check
-I- Lids Check finished successfully

---------------------------------------------
Links Check
-I- Links Check finished successfully

---------------------------------------------
Subnet Manager
-I- SM information retrieving finished successfully

-I- Subnet Manager Check finished successfully

---------------------------------------------
Port Counters
-I- Resetting ... 19/19 nodes (1/1 Switches & 18/18 CA-s) reseted.
-I- Ports counters reseting finished successfully

-I- Retrieving ... 19/19 nodes (1/1 Switches & 18/18 CA-s) retrieved.
-I- Ports counters retrieving finished successfully

-I- Going to sleep for 30 seconds until next counters sample
-I- Time left to sleep ... 1 seconds..

-I- Retrieving ... 19/19 nodes (1/1 Switches & 18/18 CA-s) retrieved.
-I- Ports counters retrieving (second time) finished successfully

-E- Ports counters value Check finished with errors
-E- lid=0x000a dev=26428 computeB2/U1/P1
        Performance Monitor counter         : Value
        symbol_error_counter                : 6372       (threshold=1)
-E- lid=0x0013 dev=26428 computeC4/U1/P1
        Performance Monitor counter         : Value
        symbol_error_counter                : 5          (threshold=1)

-E- Ports counters Difference Check (during run) finished with errors
-E- computeB2/U1/P1 - "symbol_error_counter" increased during the run (difference
value=6363,difference allowed threshold=1)
-E- computeC4/U1/P1 - "symbol_error_counter" increased during the run (difference
value=5,difference allowed threshold=1)

-E- BER Check finished with errors
-E- computeB2/U1/P1 - BER exceeds the threshold in port = computeB2/U1/P1(BER
value=4.938338e-09, threshold=1.000000e-12)
-E- computeC4/U1/P1 - BER exceeds the threshold in port = computeC4/U1/P1(BER
value=3.880511e-12, threshold=1.000000e-12)

---------------------------------------------
Nodes Information
-I- Retrieving ... 19/19 nodes (1/1 Switches & 18/18 CA-s) retrieved.
-I- Nodes information retrieving finished successfully

-E- FW Check finished with errors
-I- Errors/Warnings list will be reported in log file
```

```
---------------------------------------------
Speed / Width checks
-I- Link Speed Check (Expected value given = 10)
-I- Links Speed Check finished successfully

-I- Link Width Check (Expected value given = 4x)
-I- Links Width Check finished successfully

---------------------------------------------
Partition Keys
-I- Retrieving ... 19/19 nodes (1/1 Switches & 18/18 CA-s) retrieved.
-I- Partition Keys retrieving finished successfully

-I- Partition Keys finished successfully

---------------------------------------------
Alias GUIDs
-I- Retrieving ... 19/19 nodes (1/1 Switches & 18/18 CA-s) retrieved.
-I- Alias GUIDs retrieving finished successfully

-I- Alias GUIDs finished successfully

---------------------------------------------
Summary
-I- Stage                    Warnings   Errors     Comment
-I- Discovery                0          0
-I- Lids Check               0          0
-I- Links Check              0          0
-I- Subnet Manager           0          0
-I- Port Counters            0          6
-I- Nodes Information        0          8
-I- Speed / Width checks     0          0
-I- Partition Keys           0          0
-I- Alias GUIDs              0          0

-I- You can find detailed errors/warnings in: /var/tmp/ibdiagnet2/ibdiagnet2.log


-I- ibdiagnet database file   : /var/tmp/ibdiagnet2/ibdiagnet2.db_csv
-I- LST file                  : /var/tmp/ibdiagnet2/ibdiagnet2.lst
-I- Subnet Manager file       : /var/tmp/ibdiagnet2/ibdiagnet2.sm
-I- Ports Counters file       : /var/tmp/ibdiagnet2/ibdiagnet2.pm
-I- Nodes Information file    : /var/tmp/ibdiagnet2/ibdiagnet2.nodes_info
-I- Partition keys file       : /var/tmp/ibdiagnet2/ibdiagnet2.pkey
-I- Alias guids file          : /var/tmp/ibdiagnet2/ibdiagnet2.aguid


[root@computeB1 ~]#

[root@c902mnx01 namechange]#
```

The tool creates log files with detailed information under /var/tmp/ibdiagnet2. For example, a request for cable information using the **--get_cable_info** parameter writes a file named ibdiagnet2.cables into this directory. Each cable is documented, as shown in Example A-2. Note that the cable is reported from both ports that are connected to.

Also note that the **-H** help option is needed to show extended options, such as **--get_cable_info**.

Example A-2 The ibdiagnet command cable information

```
-----------------------------------------------------
Port=23 Lid=0x0003 GUID=0x0002c9030071d380 Port Name=ibswC1:SX90Y3245/U1/P23
-----------------------------------------------------
Vendor: Mellanox
OUI: 0x2c9
PN: 90Y3814
SN: 2381424T00V
Rev: A1
Length: 3 m
Type: Copper cable- unequalized
SupportedSpeed: SDR/DDR/QDR


-----------------------------------------------------
Port=1 Lid=0x0013 GUID=0x0002c9030000a7a7d Port Name=computeC3/U1/P1
-----------------------------------------------------
Vendor: Mellanox
OUI: 0x2c9
PN: 90Y3814
SN: 2381424T00V
Rev: A1
Length: 3 m
Type: Copper cable- unequalized
SupportedSpeed: SDR/DDR/QDR
```

See also tests using **ib_send_bw** and **ib_read_bw** commands, which are described in the following section.

Examples of ib_send_bw, ib_read_bw, and ib_read_lat

These tools enable tests between node pairs to check for bandwidth and latency.

The following examples show a run of send/read tests using a node pair that has one partner, computeB1, which is known to produce increasing symbol error counters.

Example A-3 shows the start of an **ib_send_bw** server on computeB1. Note that the server stops automatically after the test is run, which is different from the behavior of **iperf** (discussed in “Example of iperf” on page 87).

Example A-3 Running ib_send_bw server on computeB1

```
[root@computeB1 ~]# ib_send_bw -F
------------------------------------------------------------------------------
-----
                    Send BW Test
 Dual-port       : OFF
```

```
 Number of qps   : 1
 Connection type : RC
 RX depth        : 600
 CQ Moderation   : 50
 Mtu             : 2048B
 Link type       : IB
 Max inline data : 0B
 rdma_cm QPs     : OFF
 Data ex. method : Ethernet
---------------------------------------------------------------------------------
-----
 local address: LID 0x0b QPN 0x00d0 PSN 000000
 remote address: LID 0x0a QPN 0x00ba PSN 000000
---------------------------------------------------------------------------------
-----
 #bytes     #iterations    BW peak[MB/sec]    BW average[MB/sec]   MsgRate[Mpps]
Conflicting CPU frequency values detected: 1596.000000 != 2793.000000
Test integrity may be harmed !
Warning: measured timestamp frequency 2800.11 differs from nominal 1596 MHz
 65536      1000           -nan               3253.54              0.052057
---------------------------------------------------------------------------------
-----
[root@computeB1 ~]#
```

Example A-4 is the log from the **ib_send_bw** client started on computeB2 and connecting to computeB1. The test does not show any problems.

Example A-4 Running ib_send_bw client on computeB2 toward computeB1

```
[root@computeB2 ~]# ib_send_bw -F computeB1
---------------------------------------------------------------------------------
-----
                    Send BW Test
 Dual-port       : OFF
 Number of qps   : 1
 Connection type : RC
 TX depth        : 300
 CQ Moderation   : 50
 Mtu             : 2048B
 Link type       : IB
 Max inline data : 0B
 rdma_cm QPs     : OFF
 Data ex. method : Ethernet
---------------------------------------------------------------------------------
-----
 local address: LID 0x0a QPN 0x00ba PSN 000000
 remote address: LID 0x0b QPN 0x00d0 PSN 000000
---------------------------------------------------------------------------------
-----
 #bytes     #iterations    BW peak[MB/sec]    BW average[MB/sec]   MsgRate[Mpps]
 65536      1000           3235.39            3235.36              0.051766
---------------------------------------------------------------------------------
-----
[root@computeB2 ~]#
```

Example A-5 shows the start of an **ib_read_bw** server process on computeB1.

Example A-5 Running ib_read_bw server on computeB1

```
[root@computeB1 ~]# ib_read_bw -F
------------------------------------------------------------------------------
-----
                    RDMA_Read BW Test
 Dual-port       : OFF
 Number of qps   : 1
 Connection type : RC
 CQ Moderation   : 50
 Mtu             : 2048B
 Link type       : IB
 Outstand reads  : 16
 rdma_cm QPs     : OFF
 Data ex. method : Ethernet
------------------------------------------------------------------------------
-----
 local address: LID 0x0b QPN 0x00cf PSN 0xf4caaa OUT 0x10 RKey 0x012600 VAddr
0x007f1fa829a000
 remote address: LID 0x0a QPN 0x00b9 PSN 0x90ef4f OUT 0x10 RKey 0x00b300 VAddr
0x007f332eed5000
------------------------------------------------------------------------------
-----
[root@computeB1 ~]#
```

Example A-6 shows the results of an **ib_read_bw** from computeB2 toward computeB1 server. Note that the throughput is drastically reduced. Example A-6 shows how the symbol errors affect the behavior of the nodes' communication.

Example A-6 Running ib_read_bw client on computeB2 toward computeB1

```
[root@computeB2 ~]# ib_read_bw -F computeB1
------------------------------------------------------------------------------
-----
                    RDMA_Read BW Test
 Dual-port       : OFF
 Number of qps   : 1
 Connection type : RC
 TX depth        : 300
 CQ Moderation   : 50
 Mtu             : 2048B
 Link type       : IB
 Outstand reads  : 16
 rdma_cm QPs     : OFF
 Data ex. method : Ethernet
------------------------------------------------------------------------------
-----
 local address: LID 0x0a QPN 0x00b9 PSN 0x90ef4f OUT 0x10 RKey 0x00b300 VAddr
0x007f332eed5000
 remote address: LID 0x0b QPN 0x00cf PSN 0xf4caaa OUT 0x10 RKey 0x012600 VAddr
0x007f1fa829a000
------------------------------------------------------------------------------
-----
```

```
 #bytes     #iterations    BW peak[MB/sec]    BW average[MB/sec]   MsgRate[Mpps]
 65536      1000           3173.06            62.07                0.000993
---------------------------------------------------------------------------------
-----
[root@computeB2 ~]#
```

It is a good practice to monitor the fabric while performing such tests. For example, a run of the output of **ibdiagnet** reveals any errors showing up during the tests. See “Example of ibdiagnet” on page 76 as an example of bad links producing bad throughput. Other options are to monitor the Unified Fabric Manager (UFM) Dashboard to check the fabric health.

GPFS related tools

A few tools are available for GPFS and delivered as source code with GPFS. A functional development environment is needed to build tools such as **nsdperf**. A leading practice is to perform the build on one server, and then keep the executable file in some shared home directory.

The **nsdperf** utility is meant to be run for testing network performance. The difference between **nsdperf** and other tools is the fact that **nsdperf** tries to generate traffic patterns matching the GPFS I/O behavior as close as possible. Unlike **iperf**, **nsdperf** performs tests in a many-to-many configuration, where sets of server and client nodes can be defined for each test.

The **gpfsperf** utility enables you to measure the real I/O performance on disk. This is useful to test the performance of the disk systems in conjunction with the network and the GPFS file system.

Disk performance should be verified in advance by using, for example, the **dd** utility. Only if **dd** or similar tools show good raw disk performance values does it makes sense to proceed with other tools, such as **gpfsperf**.

In addition, GPFS provides commands that provide for checks, for example **mmdiag**.

Example of nsdperf

The **nsdperf** utility is included as source code with the GPFS installation. The sources can be found in the `/usr/lpp/mmfs/samples/net` directory.

Note: Just running **make** works, but the resulting binary does not include Remote Direct Memory Access (RDMA) support. To add the RDMA support to your binary, use the following command:

```
g++ -O2 -DRDMA -o nsdperf -lpthread -lrt -libverbs -lrdmacm nsdperf.C
```

After the binary is created, it might be run on the nodes. It is useful to create some script framework around it. One example of how this can be done is found on the IBM developerWorks website:

https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20%28GPFS%29/page/Testing%20network%20performance%20with%20nsdperf

To create the script framework, follow these steps:

1. Create one file named `testnodes`, containing all of the test node names, one name on each line.
2. Create one file named `servers` and one file named `clients`, which contain one node name on each line. These are subsets of the `testnodes` file.
3. Use the **gencommand** script to generate a command file from this information.
4. Use the script control to control and check the **nsdperf** server processes. The command control starts the server processes.

Because **nsdperf** relies on the network and produces high traffic, it is suggested that you monitor the network with either **ibdiagnet** or the UFM dashboard to determine if there are network faults while **nsdperf** runs.

A run without RDMA is shown in Example A-8. The command file used for this run is shown in Example A-7.

Example A-7 Listing of the nsdperf command file (non-RDMA)

```
server computeB5
server computeB6
client computeB1
client computeB2
client computeB3
client computeB4
tt 120
test
quit
```

The resulting test run is shown in Example A-8.

Example A-8 The nsdperf command run without RDMA

```
[root@computeB1 nsdperf]# ./nsdperf -i commands
Connected to computeB5
Connected to computeB6
Connected to computeB1
Connected to computeB2
Connected to computeB3
Connected to computeB4
Test time set to 120 seconds
4-2 write 184 MB/sec (43.8 msg/sec), cli 1rv 1%, time 120, buff 4194304
4-2 read 237 MB/sec (56.4 msg/sec), cli 1rv 1%, time 120, buff 4194304
[root@computeB1 nsdperf]#
```

A run with RDMA is shown in Example A-10 on page 85. For this run, the commands file has been changed to include the **rdma on** command, as shown in Example A-9.

Example A-9 Listing of the nsdperf command file (using RDMA)

```
server computeB5
server computeB6
client computeB1
client computeB2
client computeB3
```

```
client computeB4
tt 120
rdma on
test
quit
```

The resulting test run is shown in Example A-10.

Example A-10 The nsdperf command run with RDMA

```
[root@computeB1 nsdperf]# ./nsdperf -i commands
Connected to computeB5
Connected to computeB6
Connected to computeB1
Connected to computeB2
Connected to computeB3
Connected to computeB4
Test time set to 120 seconds
RDMA is now on
4-2 write 7140 MB/sec (1700 msg/sec), cli 1rv 1%, time 120, buff 4194304, RDMA
4-2 read 5690 MB/sec (1360 msg/sec), cli 3rv 1%, time 120, buff 4194304, RDMA
[root@computeB1 nsdperf]#
```

For completeness, the scripts used for this example are listed in the following examples. These are the scripts from the website referenced previously, with a change to the `rootdir` in the control script.

Example A-11 shows the control script that is used to control the **`nsdperf`** processes.

Example A-11 The control script for running nsdperf

```
#!/bin/bash

rootdir="/u/herbertm//nsdperf"

if [ "${1}" == "start" ]; then
  for i in `cat ${rootdir}/testnodes`
  do
    echo "Starting server on ${i}"
    ssh ${i} "${rootdir}/nsdperf -s < /dev/null > /dev/null 2>&1 &"
  done
elif [ "${1}" == "status" ]; then
  for i in `cat ${rootdir}/testnodes`
  do
    echo -n "${i} - "
    nprocs=`ssh ${i} "ps -ef | grep nsdperf | egrep -v grep | wc -l " `
    if (($nprocs == 0 )); then
      echo The server is not running: $nprocs
    elif (($nprocs == 1 )); then
      echo The server is running: $nprocs
    elif (($nprocs > 1 )); then
      echo Error - there are $nprocs servers running.
    fi
  done
elif [ "${1}" == "stop" ]; then
  for i in `cat ${rootdir}/testnodes`
```

```
  do
    echo Stopping server on ${i}
    ssh ${i} "ps -ef | grep nsdperf | awk '{ print \$2 }' | xargs kill -9 "
  done
fi
```

Example A-12 shows the **gencommand** script that can be used to create the **nsdperf** command file.

Example A-12 The gencommand script for generating the nsdperf command file

```
#!/bin/bash

echo "" > commands

for i in `cat servers`
do
  echo  server ${i} >> commands
done

for i in `cat clients`
do
  echo  client ${i} >> commands
done

echo tt 120 >> commands
echo rdma on >> commands
echo test >> commands
echo quit >> commands
```

An example run of **daxpy** using the **hcrun** command from the CHC toolkit is shown in 5.3.7, “The run_daxpy check” on page 51.

Example of gpfsperf

The **gpfsperf** utility is included as source code with the GPFS installation. The sources can be found in the `/usr/lpp/mmfs/samples/perf` directory. To build the non-MPI version, just run **make** and **gpfsperf** is built.

The makefile provides the ability to create a Message Passing Interface (MPI) version of the **gpfsperf** utility as well. In this example, **gpfsperf** is built using the PE. This requires you to have the PE Runtime Edition installed, and a path set to the `mpicc` executable file. Run **make** with the correct parameters, as shown in Example A-13, to create the `gpfsperf-mpi` executable file.

Example A-13 Creating the gpfsperf-mpi executable file

```
[root@computeB6 perf]# export PATH=$PATH:/opt/ibmhpc/pecurrent/ppe.poe/bin
[root@computeB6 perf]# make gpfsperf-mpi
cc -c -O -DGPFS_LINUX  -I/usr/lpp/mmfs/include irreg.c
mpicc -c -O -DGPFS_LINUX  -I/usr/lpp/mmfs/include -DMULTI_NODE gpfsperf.c -o
gpfsperf-mpi.o
mpicc gpfsperf-mpi.o irreg.o -O -DGPFS_LINUX  -lmpich -lpthread -lrt -lgpfs -o
gpfsperf-mpi
[root@computeB6 perf]#
```

A test run with the MPI executable file using a `hostlist` consisting of five compute nodes is shown in Example A-14.

Example A-14 Test run with MPI

```
-bash-4.1$ export MP_HOSTFILE=hostlist
-bash-4.1$ export MP_PROCS=`cat hostlist |wc -l`
-bash-4.1$ cat hostlist
computeB1
computeB2
computeB3
computeB5
computeB6
-bash-4.1$ poe /u/herbertm/gpfsperf/gpfsperf-mpi read strided /fsB/testfile
/u/herbertm/gpfsperf/gpfsperf-mpi read strided /fsB/testfile
  recSize 256K nBytes 999M fileSize 999M
  nProcesses 5 nThreadsPerProcess 1
  file cache flushed before test
  not using data shipping
  not using direct I/O
  offsets accessed will cycle through the same file segment
  not using shared memory buffer
  not releasing byte-range token after open
    Data rate was 107625.91 Kbytes/sec, thread utilization 0.911
-bash-4.1$
```

Network-related tools

There are many tools available for testing network performance.

Example of iperf

The **iperf** utility enables you to test network throughput using client/server connection pairs. Many options are available to run the tool. It is easily built and used for quick tests of network performance. The tool can be obtained as source code from the following website:

http://sourceforge.net/projects/iperf/

The executable file might be put, for example, into `/usr/local/sbin` and then run in client/server relationships. This section shows short examples. For more details, see the **iperf** documentation. For testing, it is useful to set a quite high TCP window size. and to use some parallelism. The examples use TCP window size 256K and four connections. Output is formatted to show MBps and test time is set to five seconds.

Example A-15 shows the server started on computeB2, and it is waiting for incoming connections from **iperf** clients. The incoming connections are shown after a client connects. In this case, the computeB1 client connects to the server. The server must be stopped manually, because it remains in a listening state after test completion.

Example A-15 Starting an iperf server on computeB2

```
[root@computeB2 ~]# iperf -s -w 262144 -fM
------------------------------------------------------------
Server listening on TCP port 5001
```

```
TCP window size: 0.50 MByte (WARNING: requested 0.25 MByte)
------------------------------------------------------------
[  4] local 10.3.1.33 port 5001 connected with 10.3.1.31 port 39690
[  5] local 10.3.1.33 port 5001 connected with 10.3.1.31 port 39691
[  6] local 10.3.1.33 port 5001 connected with 10.3.1.31 port 39692
[  7] local 10.3.1.33 port 5001 connected with 10.3.1.31 port 39693
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 5.0 sec   141 MBytes  28.2 MBytes/sec
[  5]  0.0- 5.0 sec   142 MBytes  28.2 MBytes/sec
[  6]  0.0- 5.0 sec   142 MBytes  28.4 MBytes/sec
[  7]  0.0- 5.0 sec   142 MBytes  28.3 MBytes/sec
[SUM]  0.0- 5.0 sec   568 MBytes   113 MBytes/sec
^C[root@computeB2 ~]#
```

Example A-16 shows the run of the **iperf** client on computeB1 for Example A-15 connecting to computeB2 by the 1 GBps Ethernet interface.

Example A-16 Starting an iperf client on ComputeB1

```
[root@computeB1 ~]# iperf -c computeB2 -w 256K -fM -t 5 -P 4
------------------------------------------------------------
Client connecting to computeB2, TCP port 5001
TCP window size: 0.50 MByte (WARNING: requested 0.25 MByte)
------------------------------------------------------------
[  5] local 10.3.1.31 port 39693 connected with 10.3.1.33 port 5001
[  3] local 10.3.1.31 port 39690 connected with 10.3.1.33 port 5001
[  4] local 10.3.1.31 port 39691 connected with 10.3.1.33 port 5001
[  6] local 10.3.1.31 port 39692 connected with 10.3.1.33 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0- 5.0 sec   142 MBytes  28.4 MBytes/sec
[  3]  0.0- 5.0 sec   141 MBytes  28.3 MBytes/sec
[  4]  0.0- 5.0 sec   142 MBytes  28.3 MBytes/sec
[  6]  0.0- 5.0 sec   142 MBytes  28.5 MBytes/sec
[SUM]  0.0- 5.0 sec   568 MBytes   113 MBytes/sec
[root@computeB1 ~]#
```

Example A-15 and Example A-16 run on a 1GBit Ethernet interface. The **iperf** command can also be run using IP over InfiniBand (IPoIB). Example A-17 again shows the server started on ComputeB2, with client ComputeB1 now connecting using IPoIB.

Example A-17 The iperf command run on computeB2

```
[root@computeB2 ~]# iperf -s -w 262144 -fM
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 0.50 MByte (WARNING: requested 0.25 MByte)
------------------------------------------------------------
[  4] local 20.1.1.33 port 5001 connected with 20.1.1.31 port 50518
[  5] local 20.1.1.33 port 5001 connected with 20.1.1.31 port 50519
[  6] local 20.1.1.33 port 5001 connected with 20.1.1.31 port 50520
[  7] local 20.1.1.33 port 5001 connected with 20.1.1.31 port 50521
[ ID] Interval       Transfer     Bandwidth
[  6]  0.0- 5.0 sec  2196 MBytes   439 MBytes/sec
[  4]  0.0- 5.1 sec  1955 MBytes   386 MBytes/sec
[  7]  0.0- 5.1 sec  1974 MBytes   387 MBytes/sec
```

```
[  5]  0.0- 5.1 sec  1866 MBytes   362 MBytes/sec
[SUM]  0.0- 5.1 sec  7990 MBytes  1552 MBytes/sec
^C[root@computeB2 ~]#
```

Example A-18 shows the corresponding output of client computeB1 for the previous IPoIB connection to computeB2.

Example A-18 The computeB1 client connecting to ib0

```
[root@computeB1 ~]# iperf -c 20.1.1.33 -w 256K -fM -t 5 -P 4
------------------------------------------------------------
Client connecting to 20.1.1.33, TCP port 5001
TCP window size: 0.50 MByte (WARNING: requested 0.25 MByte)
------------------------------------------------------------
[  6] local 20.1.1.31 port 50521 connected with 20.1.1.33 port 5001
[  3] local 20.1.1.31 port 50518 connected with 20.1.1.33 port 5001
[  4] local 20.1.1.31 port 50519 connected with 20.1.1.33 port 5001
[  5] local 20.1.1.31 port 50520 connected with 20.1.1.33 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0- 5.0 sec  2196 MBytes   439 MBytes/sec
[  3]  0.0- 5.1 sec  1955 MBytes   386 MBytes/sec
[  6]  0.0- 5.1 sec  1974 MBytes   388 MBytes/sec
[  4]  0.0- 5.1 sec  1866 MBytes   363 MBytes/sec
[SUM]  0.0- 5.1 sec  7990 MBytes  1554 MBytes/sec
[root@computeB1 ~]#
```

Throughput values are much higher than with the Ethernet-based connection. Note also that the throughput using IPoIB is lower than the maximum bandwidth that can be expected from the InfiniBand link. This is caused by the resource usage that the IPoIB protocol implies.

Disk benchmarks

This section describes tools that can be used to benchmark file system performance.

The **IOR** utility is useful for testing parallel file system I/O performance. The utility enables you to use different application programming interfaces (APIs), such as Portable Operating System Interface (POSIX) or MPI-I/O.

The **mdtest** utility is useful for testing metadata performance of the file system. It performs create, read, write, stat, and delete operations on files and directories, and writes out statistics about the obtained results.

Both of these utilities, **IOR** and **mdtest**, must be built and run under MPI control. The following examples have been built with PE.

The tests are run on three Network Shared Disk (NSD) client nodes on ClusterB GPFS file system fsB using the following `hostfile` contents, as shown in Example A-19.

Example A-19 MPI hostfile used for disk benchmarks

```
computeB1
computeB2
computeB3
```

Example of IOR

The **IOR** benchmark can be downloaded at the following site:

http://sourceforge.net/projects/ior-sio/

Note that the **IOR** benchmark has moved from sourceforge to github. To build the latest versions go to the following website:

https://github.com/chaos/ior

The build process for the previous version (2.10.3) is simple. Change into the unpacked **IOR** directory and issue the following commands:

```
export PATH=$PATH:/opt/ibmhpc/pecurrent/ppe.poe/bin
gmake posix
```

These steps ensure that the mpicc script is found and builds the POSIX version. For more detailed information, see the readme file provided with the package.

For the current version (3.0.1) from github, automake is needed. First clone the git repository:

```
git clone https://github.com/chaos/ior.git
```

Change into the directory and run **bootstrap** to build the configure script, if not present. Then run **configure** to prepare the environment and build the executable file:

```
export PATH=$PATH:/opt/ibmhpc/pecurrent/ppe.poe/bin
./configure LIBS=/usr/lpp/mmfs/lib/libgpfs.so
make
```

These steps ensure that the mpicc script is found, and that the linker does not return errors about undefined references to `gpfs_fcntl` during the `make` process. See the `readme` file provided with the tool.

IOR version 3.0.1 supports some GPFS-specific features (for example: **gpfsHintAccess**) that are not described in this section.

Because **IOR** relies on the network, it is suggested that you monitor the network with either **ibdiagnet** or the UFM dashboard to determine if there are network faults while **IOR** runs. With the UFM dashboard, in addition to looking for error events, you can look for congestion using the traffic monitor.

Example A-20 shows a sample run of **IOR** using three hosts, as shown in Example A-19. This test is started on one of the compute nodes, and parameters are passed by command line.

Example A-20 IOR example with command-line parameters

```
-bash-4.1$ export MP_HOSTFILE=/u/herbertm/hostfile
-bash-4.1$ poe /u/herbertm/IOR/src/C/IOR -w -r -k -Z -b 1G -i 5 -t 2M -o
/fsB/iortest
IOR-2.10.3: MPI Coordinated Test of Parallel I/O

Run began: Thu Oct 10 03:08:48 2013
Command line used: /u/herbertm/IOR/src/C/IOR -w -r -k -Z -b 1G -i 5 -t 2M -o
/fsB/iortest
Machine: Linux computeB1

Summary:
        api                = POSIX
        test filename      = /fsB/iortest
```

```
        access              = single-shared-file
        ordering in a file = sequential offsets
        ordering inter file=random task offsets >= 1, seed=0
        clients             = 3 (1 per node)
        repetitions         = 5
        xfersize            = 2 MiB
        blocksize           = 1 GiB
        aggregate filesize = 3 GiB

Operation  Max (MiB)  Min (MiB)  Mean (MiB)   Std Dev  Max (OPs)  Min (OPs)  Mean
(OPs)   Std Dev  Mean (s)
---------  ---------  ---------  ----------   -------  ---------  ---------
----------   -------  --------
write         178.65     144.48      160.96     12.86      89.32      72.24
80.48      6.43  19.20773   EXCEL
read          232.23     145.62      177.85     32.11     116.11      72.81
88.92     16.06  17.79651   EXCEL

Max Write: 178.65 MiB/sec (187.33 MB/sec)
Max Read:  232.23 MiB/sec (243.51 MB/sec)

Run finished: Thu Oct 10 03:11:57 2013
-bash-4.1$
```

Example A-21 uses a command file that can be passed to **IOR**. The command file replaces the command-line options that have been passed in Example A-20 on page 90. The file also enables you to run multiple tests in one file as shown in Example A-21. This is using **IOR** version 3.0.1 and specifies the new GPFS flags.

Example A-21 Command file

```
IOR START
 writeFile=1
 readFile=1
 keepFile=1
 randomOffset=1
 blockSize=1G
 repetitions=2
 transferSize=2M
 testFile=/fsB/newtest
 gpfsReleaseToken=0
 gpfsHintAccess=0
RUN
 repetitions=1
 gpfsReleaseToken=1
 gpfsHintAccess=1
RUN
IOR STOP
```

The result of a run using the command file is shown in Example A-22.

Example A-22 Run result using the command file

```
-bash-4.1$ poe /u/herbertm/IOR3/src/ior -f /u/herbertm/ior3cmd
IOR-3.0.1: MPI Coordinated Test of Parallel I/O

Began: Thu Oct 17 11:14:10 2013
Command line used: /u/herbertm/IOR3/src/ior -f /u/herbertm/ior3cmd
Machine: Linux computeB2

Test 0 started: Thu Oct 17 11:14:10 2013
Summary:
        api                = POSIX
        test filename      = /fsB/newtest
        access             = single-shared-file
        ordering in a file = random offsets
        ordering inter file= no tasks offsets
        clients            = 3 (1 per node)
        repetitions        = 2
        xfersize           = 2 MiB
        blocksize          = 1 GiB
        aggregate filesize = 3 GiB

access    bw(MiB/s)  block(KiB) xfer(KiB)  open(s)    wr/rd(s)   close(s)
total(s)   iter
------    ---------  ---------- ---------  --------   --------   --------
--------   ----
write     228.39     1048576    2048.00    3.11       13.43      4.86       13.45
0
read      138.09     1048576    2048.00    0.000228   22.25      7.61       22.25
0
write     146.92     1048576    2048.00    0.485971   20.91      0.277916   20.91
1
read      160.04     1048576    2048.00    0.000438   19.19      4.78       19.19
1

Max Write: 228.39 MiB/sec (239.49 MB/sec)
Max Read:  160.04 MiB/sec (167.82 MB/sec)

Test 1 started: Thu Oct 17 11:15:32 2013
Summary:
        api                = POSIX
        test filename      = /fsB/newtest
        access             = single-shared-file
        ordering in a file = random offsets
        ordering inter file= no tasks offsets
        clients            = 3 (1 per node)
        repetitions        = 1
        xfersize           = 2 MiB
        blocksize          = 1 GiB
        aggregate filesize = 3 GiB

access    bw(MiB/s)  block(KiB) xfer(KiB)  open(s)    wr/rd(s)   close(s)
total(s)   iter
```

```
------    ---------  ---------- ---------  --------   --------   --------
--------   ----
write     68.84      1048576    2048.00    0.072261   44.62      9.67       44.62
0
read      152.41     1048576    2048.00    0.001203   20.16      3.32       20.16
0

Max Write: 68.84 MiB/sec (72.19 MB/sec)
Max Read:  152.41 MiB/sec (159.81 MB/sec)

Summary of all tests:
Operation   Max(MiB)   Min(MiB)  Mean(MiB)     StdDev    Mean(s) Test# #Tasks tPN
reps fPP reord reordoff reordrand seed segcnt blksiz xsize aggsize API RefNum
write         228.39     146.92     187.66      40.74   17.17979 0 3 1 2 0 0 1 0 0
1 1073741824 2097152 3221225472 POSIX 0
read          160.04     138.09     149.06      10.98   20.72093 0 3 1 2 0 0 1 0 0
1 1073741824 2097152 3221225472 POSIX 0
write          68.84      68.84      68.84       0.00   44.62437 1 3 1 1 0 0 1 0 0
1 1073741824 2097152 3221225472 POSIX 0
read          152.41     152.41     152.41       0.00   20.15649 1 3 1 1 0 0 1 0 0
1 1073741824 2097152 3221225472 POSIX 0

Finished: Thu Oct 17 11:16:37 2013
-bash-4.1$
```

Example of mdtest

The code for **mdtest** can be downloaded at the following website:

http://sourceforge.net/projects/mdtest/

The version used for this section is 1.9.1. Example A-23 shows a simple example of running **mdtest** on clusterB. The recursive directory depth is set to 4. The branch is set to 6, so that in each directory, 6 subdirectories are recursively created (**-b** parameter) until depth of 4 (**-z** parameter) is reached. Within each of these directories, each node then creates 5 (**-I** parameter) empty directories and 5 (**-i** parameter) empty files, which are used for testing.

The number of used nodes can grow quickly. The number of files and directories that will be created can easily be precalculated with the following formula:

```
#nodes * I * (1- b exp(z+1) / (1-b))
```

For example, using depth z=5, I=10, b=6 and running **mdtest** on 1000 nodes produces 93 million files and 93 million directories.

Example A-23 An mdtest run on clusterB

```
-bash-4.1$ export MP_HOSTFILE=/u/herbertm/hostfile
-bash-4.1$ export MP_PROCS=`cat \`echo $MP_HOSTFILE\`|wc -l`
-bash-4.1$ poe /u/herbertm/mdtest-1.9.1/mdtest -I 5 -z 4 -b 6 -i 5
-- started at 10/10/2013 05:57:37 --

mdtest-1.9.1 was launched with 3 total task(s) on 3 node(s)
Command line used: /u/herbertm/mdtest-1.9.1/mdtest -I 5 -z 4 -b 6 -i 5
Path: /fsB
FS: 273.5 GiB   Used FS: 1.9%   Inodes: 0.1 Mi   Used Inodes: 2.9%
```

```
3 tasks, 23325 files/directories

SUMMARY: (of 5 iterations)
   Operation                     Max            Min           Mean        Std Dev
   ---------                     ---            ---           ----        -------
   Directory creation:         458.857        378.426        401.618         29.997
   Directory stat    :        1845.778       1617.817       1677.453         85.202
   Directory removal :         530.626        449.805        506.142         29.322
   File creation     :         638.294        550.174        602.203         29.341
   File stat         :        2222.771       1328.575       1685.229        336.146
   File read         :        2334.903       1517.298       1943.249        277.179
   File removal      :         541.109        468.839        513.678         27.210
   Tree creation     :        2573.000        460.983       1512.702        708.913
   Tree removal      :         258.909        191.060        214.722         25.638

-- finished at 10/10/2013 06:17:29 --
-bash-4.1$
```

Node-specific tools

There are some tools that are useful for checking the node health status. Especially after repairing actions, these tests turn out to be useful before bringing a node into production.

The **stream** benchmark enables you to measure sustainable memory bandwidth and helps to detect any issues with installed dual inline memory modules (DIMMs) that might influence performance of a node while running jobs. The National Energy Research Scientific Computing Center (NERSC) describes STREAM as *a simple, synthetic benchmark designed to measure sustainable memory bandwidth (in MBps), and a corresponding computation rate for four simple vector kernels*:

https://www.nersc.gov/systems/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks/stream/

In addition, memory allocation tests are useful to check, if enough memory is available and really usable on a node.

The **dgemm** and **daxpy** tests enables you to test compute performance. For this test, it is important to have a reasonable performance value that can be assumed as a baseline, to determine success or failure of a given test on a node.

The **linpack** benchmark also enables you to detect any issues with compute performance, and can be run as single-node **linpack** to check the capabilities of a compute node.

Example of stream

The stream program code can be downloaded from the following website:

http://www.cs.virginia.edu/stream/ref.html

The source code and makefile are available on the following website:

http://www.cs.virginia.edu/stream/FTP/Code

This version can be built from within the source directory using the makefile. Run the following command:

```
make stream_c.exe
```

In addition, an Open Multi-Processing (OpenMP) API source code version, `stream_omp.c`, is available on the following website:

http://www.cs.virginia.edu/stream/FTP/Code/Versions

The OpenMP version can be built using the following command:

```
gcc -fopenmp -D OPENMP stream_omp.c -o stream_omp.exe
```

The version used in the examples is version 5.10.

Example A-24 shows the **stream** run for the OpenMP version that has been built for this test. Note that the environment variable for the number of OpenMP threads to be used is set in advance.

Example A-24 Stream run with the OpenMP version

```
[root@computeA1 stream]# export OMP_NUM_THREADS=16
[root@computeA1 stream]# ./stream_omp.exe
-------------------------------------------------------------
This system uses 8 bytes per DOUBLE PRECISION word.
-------------------------------------------------------------
Array size = 200000000, Offset = 0
Total memory required = 4577.6 MB.
Each test is run 10 times, but only
the *best* time for each is used.
-------------------------------------------------------------
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 56187 microseconds.
   (= 56187 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
```

```
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       37365.3211       0.0866       0.0856       0.0873
Scale:      36643.0770       0.0883       0.0873       0.0889
Add:        38379.4300       0.1264       0.1251       0.1274
Triad:      37416.0836       0.1292       0.1283       0.1297
-------------------------------------------------------------
Solution Validates
-------------------------------------------------------------
[root@computeA1 stream]#
```

Example A-25 shows the same run, but in this case the threads' affinity to CPUs has been set. This is done by setting the **GOMP_CPU_AFFINITY** environment variable.

Example A-25 Run with the threads' affinity to CPU set

```
[root@computeA1 stream]# export OMP_NUM_THREADS=16
[root@computeA1 stream]# export GOMP_CPU_AFFINITY=0-31
[root@computeA1 stream]# ./stream_omp.exe
-------------------------------------------------------------
This system uses 8 bytes per DOUBLE PRECISION word.
-------------------------------------------------------------
Array size = 200000000, Offset = 0
Total memory required = 4577.6 MB.
Each test is run 10 times, but only
the *best* time for each is used.
-------------------------------------------------------------
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
Number of Threads requested = 16
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 48813 microseconds.
   (= 48813 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       43772.0268       0.0732       0.0731       0.0735
```

```
Scale:      43176.8176       0.0743       0.0741       0.0746
Add:        47566.6015       0.1016       0.1009       0.1019
Triad:      47228.8318       0.1020       0.1016       0.1023
-------------------------------------------------------------
Solution Validates
-------------------------------------------------------------
[root@computeA1 stream]#
```

Example of dgemm

The **dgemm** utility is included in the PE installation in the following directory:

`/opt/ibmhpc/pecurrent/systemcheck/bin`

The contents from this directory should be copied to a directory for which the user running the tool is the owner. In this case. `/tmp/systemcheck/bin` is used. This is also the directory where the CHC toolset places the `systemcheck` utilities.

Example A-26 shows the run on a single node. The tool is run under PE, so a `hostlist` must be specified. In this case, the `hostlist` only contains a single node.

Example A-26 Running dgemm on a single node

```
bash-4.1$ cat /u/herbertm/hostlist
computeA2
-bash-4.1$ export MP_HOSTFILE=/u/herbertm/hostlist
-bash-4.1$ ./run.dgemm -V
computeA2:   Cores=16 perf (GFlop)=  368
PERFORMANCE_SUCCESS: EXPECTED_PERF 311 : computeA2 368 GF
-bash-4.1$
```

A wrapper script named `do_poe_dgemm` is provided. This either assumes that **MP_HOSTFILE** is already exported with a list of nodes, or that the `host.list` file is present in the current directory and contains the list of nodes.

Example A-27 shows the run of this wrapper script.

Example A-27 Running dgemm using the wrapper script

```
-bash-4.1$ ./do_poe_dgemm
computeA2:   Cores=16 perf (GFlop)=  368
PERFORMANCE_SUCCESS: EXPECTED_PERF 311 : computeA2 368 GF
-bash-4.1$
```

An example run of **dgemm** using the **hcrun** command from the CHC toolkit is shown in 5.3.8, “The run_dgemm check” on page 52.

Example of daxpy

The **daxpy** utility is included in the PE installation in the following directory:

`/opt/ibmhpc/pecurrent/systemcheck/bin`

From there, it can be run on a single node, as shown in Example A-28. The example has been run under root, because the daxpy test does not use MPI and PE. Daxpy is an OpenMP program, so the `OMP_NUM_THREADS` variable and CPU affinity settings are important. The `run.daxpy` script tries to set these variables automatically.

Example A-28 Running daxpy on a single node

```
[root@computeA2 bin]# ./run.daxpy -V
estimated EXPECTED_PERF 61318
computeA2: Expected_DAXPY=61318  OMP_NUM_THREADS=16 perf (DAXPY BW)=  64538
computeA2: PERFORMANCE_SUCCESS: 64538 GF;
[root@computeA2 bin]#
```

For `run.daxpy`, a wrapper named `poe.run.daxpy` also exists, similar to that for **dgemm**. This launches `run.daxpy` under PE control. It either assumes that **MP_HOSTFILE** is already exported with a list of nodes, or that the host.list files is present in the current directory and contains the list of nodes.

Example A-29 shows the run of **do_poe_daxpy** using a `hostlist` containing computeA2. This demonstrates a failing example.

Example A-29 Running do_poe_daxpy using a hostlist in computeA2

```
-bash-4.1$ export MP_HOSTFILE=/u/herbertm/hostlist
-bash-4.1$ ./do_poe_daxpy
computeA2: PERFORMANCE_FAILURE: 19906 MB/s;
computeA2: Expected_DAXPY=61500  OMP_NUM_THREADS=16 perf (DAXPY BW)=  19906
computeA2: PERFORMANCE_FAILURE: 19906 MB/s;
-bash-4.1$
```

An example run of **daxpy** using the **hcrun** command from the CHC toolkit is shown in 5.3.7, “The run_daxpy check” on page 51.

IBM HPC Central

This section provides a short overview of the available IBM sites for service packs, links to important sites such as the GPFS forum, and more. The primary entry point to these resources is the IBM HCP Central website.

The IBM HPC Central site is at the following website:

https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Welcome%20to%20High%20Performance%20Computing%20%28HPC%29%20Central

Service Pack information can be obtained from the following site:

http://ibm.co/1eFt9bN

B

Tools and commands outside of the toolkit

This appendix provides a description of helpful tools and commands outside of the toolkit. However, some of them are partly included in the toolkit.

This appendix provides information about the following topics:

- Remote management access
- Firmware versions
- Light-emitting diode status
- Baseboard firmware
- Component firmware
- UEFI settings [System x only]

Remote management access

Some of the following checks need access to the remote management adapter of the compute node. On System x, this is called the integrated management module (IMM or IMMv2) or the advanced management module (AMM) in a BladeCenter environment. The IBM Power Systems use either the Hardware Management Console (HMC) or the flexible service processor (FSP).

Example B-1 checks the access and power state of all the nodes in nodegroup f01.

Example B-1 Remote management access

```
# rpower f01 state|xdshcoll

====================================
c933f01x31,c933f01x33,c933f01x35,c933f01x37,c933f01x39,c933f01x41
====================================
on
```

Light-emitting diode status

After verification that the remote access to the baseboard management controller (BMC) works, verify that there are no critical errors on the compute nodes. The status of the error light-emitting diode (LED) can be checked with the **`rvitals`** command, as shown in Example B-2.

Example B-2 Error LED

```
[System x]
rvitals f01 leds|xcoll
====================================
c933f01x35,c933f01x33,c933f01x37
====================================
LED 0x0000 (Fault) active to indicate system error condition.
LED 0012 (PS) active to indicate Sensor 0x71 (Power Supply 2) error.

====================================
c933f01x39,c933f01x41,c933f01x31
====================================
No active error LEDs detected
```

In Example B-2, there are three nodes with an active fault LED. The message (in this case `Power Supply 2 error`) should be investigated, because this might affect the cluster.

The **`hcrun`** command from the Cluster Health Check (CHC) toolkit provides the same functionality as shown in 5.3.1, “The leds check” on page 48.

Firmware versions

Firmware versions can have an important effect on the functionality or performance of a cluster. The versions of the baseboard firmware, such as Unified Extensible Firmware Interface (UEFI) and IMM, should be the same on all models of the same kind throughout the cluster. It is essential that the firmware in use should be at the latest good known state, which is not necessarily the “latest and finest” version.

Interdependencies must be taken into account. Firmware levels must also fit to the respective device driver versions. This is necessary to have a supported environment, but it should also increase the stability of your cluster.

Baseboard firmware

The firmware levels of the baseboard are usually easy to determine. Example B-3 shows where the first **rinv** command gets the compute node model and, based on that, retrieves the firmware version.

Example B-3 Model and firmware versions (System x)

```
# rinv f01 model

====================================
c933f01x31,c933f01x33,c933f01x35,c933f01x37
====================================
System Description: System x3650 M2
System Model/MTM: 7947AC1

====================================
c933f01x39,c933f01x41
====================================
System Description: System x3650 M3
System Model/MTM: 7945AC1

# rinv c933f01x31,c933f01x33,c933f01x35,c933f01x37 firmware|xdshcoll
====================================
c933f01x31,c933f01x33,c933f01x35,c933f01x37
====================================
BMC Firmware: 1.35 (YUOOE9B 2012/11/29 09:14:45)
UEFI Version: 1.16 (D6E158A 2012/11/26)
```

The **hcrun** command from the CHC toolkit can check the firmware, too (see 5.3.5, “The firmware check” on page 50).

Component firmware

The actual firmware level of a component, such as a network or InfiniBand adapter, might be more difficult to determine. Linux does not provide a general approach to get the information about a Peripheral Component Interconnect (PCI) adapter. Table B-1 lists a few commands that help you gather the information.

Table B-1 Collecting firmware levels

Platform (System x or Power Systems)	Component	Command	Result
p / x	InfiniBand host channel adapter (HCA), Mellanox	ibstat	... Firmware version: 2.9.1000 ...
p / x	Ethernet adapter	ethtool -i <device>	... firmware-version: 5.10-2 ...

Platform (System x or Power Systems)	Component	Command	Result
p	Disks, RAID controller, System firmware	lsmcode -A	

UEFI settings [System x only]

Depending on the type or model of compute nodes and the cluster they form, there might be suggested settings in the UEFI of the nodes. The IBM Advanced Settings Utility (ASU) tool is used to show or change settings without entering the UEFI setup from a graphical remote console. The IBM Extreme Cloud Administration Toolkit (xCAT) provides a front end to the ASU tool that enables parallel access to all nodes in the cluster. This requires the Red Hat Package Manager (RPM) version of ASU to be installed on the xCAT management server.

Example B-4 shows four query settings from a cluster with System x 3650 M2/M3 nodes.[1] A batch file containing all of the commands is created, and afterward the **pasu** tool is called with the file shown in Example B-4 as input.

Example B-4 Query settings

```
# cat /tmp/asu.batch
show uEFI.ProcessorHyperThreading
show uEFI.OperatingMode
show uEFI.QPISpeed
show uEFI.PatrolScrub

# pasu -l USERID -p PASSWORD -b /tmp/asu.batch f01 |xcoll
====================================
ib
====================================
Batch mode start.
[show uEFI.ProcessorHyperThreading]
uEFI.ProcessorHyperThreading=Enable

[show uEFI.OperatingMode]
uEFI.OperatingMode=Custom Mode

[show uEFI.QPISpeed]
uEFI.QPISpeed=Max Performance

[show uEFI.PatrolScrub]
uEFI.PatrolScrub=Disable

Batch mode competed successfully.
```

IBM Parallel Operating Environment verification

Some of the tests for the CHC require a working IBM Parallel Operating Environment (POE) installation in the cluster. The `ppe_rte_samples` package delivers a number of small programs that can be used to verify if POE is functioning correctly. The samples are installed in `/opt/ibmhpc/pecurrent/ppe.samples`. They are accompanied by a `readme` file with further instructions for building and running the tools.

[1] Unfortunately the parameter names can vary from model to model , and between firmware versions.

Installation verification procedure

The installation verification procedure (IVP) is the install verification for POE. IVP checks for the existence of some files, and that files in `/etc` contain the correct information. IVP then compiles[2] and runs a small program where a task sends a message to all other tasks. This program should exit with an error code of `0`.

Note: It is suggested to run the check with a non-root user. Also check that the user's public key is distributed to all other nodes (including the actual host), so that password-free access for Secure Shell (SSH) sessions is possible.

Example B-5 shows a successful verification run. The compiler warnings can be ignored.

Example B-5 IVP run

```
poeuser@computeA2:/opt/ibmhpc/pecurrent/ppe.samples/ivp$ ./ivp.linux.script
Verifying the existence of the Binaries
Partition Manager daemon /etc/pmdv13 is executable
PE RTE files seem to be in order
Compiling the ivp sample program
Output files will be stored in directory /tmp/ivp3522
ivp.c: In function ‘main’:
ivp.c:42: warning: incompatible implicit declaration of built-in function ‘strlen’
ivp.c:57: warning: incompatible implicit declaration of built-in function ‘exit’
ivp.c:94: warning: incompatible implicit declaration of built-in function ‘exit’
Creating host.list file for this node
Setting the required environment variables
Executing the parallel program with 2 tasks

SHM total segment size=15885440 num_common_tasks=2 slot_per_task=512
slot_size=12416 slot_data_size=12288 slot_offset=3171456
SHM total segment size=15885440 num_common_tasks=2 slot_per_task=512
slot_size=12416 slot_data_size=12288 slot_offset=3171456
PAMI job ID for this job is: 1916227334
64bit (MPI over PAMI) ppe_rcot MPICH2 library was compiled on Wed Aug 14 14:49:38
2013

POE IVP: running as task 1 on node computeA2
POE IVP: running as task 0 on node computeA2
POE IVP: there are 2 tasks running
POE IVP: all messages sent
POE IVP: task   1 received <POE IVP Message Passing Text>

Parallel program ivp.out return code was 0


If the test returns a return code of 0, POE IVP
 is successful.  To test message passing,
 run the tests in /opt/ibmhpc/pe13xx/ppe.samples/poetest.bw and poetest.cast
 To test threaded message passing,
 run the tests in /opt/ibmhpc/pe13xx/ppe.samples/threads
End of IVP test
```

[2] The script only checks for the existence of **`mpcc`**.

At the end of the IVP run, there are additional tests mentioned. These tests are looked at in the following sections.

The POE tests

There are two POE tests provided that test the communication between nodes.

The poetest.bw test

This POE test measures the bandwidth between two nodes. These two nodes need to be specified in a file named `host.list` in the working directory. Before running **`bw.run`** on one of the nodes, the binary needs to be compiled on the participating nodes.

A successful run is provided in Example B-6.

Example B-6 POE test measures

```
poeuser@computeB1:/opt/ibmhpc/pecurrent/ppe.samples/poetest.bw$ cat host.list
computeB1
computeB2
poeuser@computeB1:/opt/ibmhpc/pecurrent/ppe.samples/poetest.bw$ ./bw.run
 hello from task           1
 hello from task           0
MEASURED BANDWIDTH =  113.096451 *10**6 Bytes/sec
```

The poetest.cast test

This test performs a broadcast from task `0` to all nodes in the cluster. Similar to the poetest.bw, a `host.list` file needs to be created. For this test, more than two nodes can be specified.

Example B-7 shows a successful run. The sequence of the resulting lines is unpredictable.

Example B-7 A poetest.cast test

```
poeuser@computeA1:/opt/ibmhpc/pecurrent/ppe.samples/poetest.cast$ cat host.list
computeA1
computeA2
computeA3
poeuser@computeA1:/opt/ibmhpc/pecurrent/ppe.samples/poetest.cast$ ./bcast.run
Broadcasting on 3 nodes
 Hello from task            2
 Hello from task            0
 BROADCAST TEST COMPLETED SUCCESSFULLY
 Hello from task            1
```

Related publications

The publications listed in this section are considered particularly suitable for providing more detailed information about the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might only be available in softcopy:

- *HPC Clusters Using InfiniBand on IBM Power Systems Servers*, SG24-7767

You can search for, view, download, or order these documents and other Redbooks, Redpapers, Web Docs, drafts, and additional materials at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- Cluster Products information center

  http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic=/com.ibm.cluster.infocenter.doc/infocenter.html

- IBM High Performance Computing (HPC) Central

  https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Welcome%20to%20High%20Performance%20Computing%20%28HPC%29%20Central

- The `stream` program code

  http://www.cs.virginia.edu/stream/ref.html

- The **`mdtest`** command

  http://sourceforge.net/projects/mdtest/

- The interoperable object reference (IOR) benchmark

  http://github.com/chaos/ior

- The **`iperf`** utility

  http://sourceforge.net/projects/iperf/

- The Leibniz Supercomputing Centre (LRZ) in Garching, Germany

  http://www.lrz.de/services/compute/supermuc/

- Official Red Hat Enterprise Linux (RHEL) documentation

  https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/index.html

- Mellanox OpenFabrics Enterprise Distribution for Linux (MLNX_OFED) documentation

  http://www.mellanox.com/page/products_dyn?product_family=26

- IBM Extreme Cloud Administration Toolkit (xCAT)

  http://xcat.sourceforge.net/

- IBM General Parallel File System (GPFS) documentation

  http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/topic/com.ibm.cluster.gpfs.doc/gpfsbooks.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

(0.2”spine)
0.17”<->0.473”
90<->249 pages

IBM High Performance Computing Cluster Health Check

Learn how to verify cluster functionality

See sample scenarios

Understand best practices

This IBM Redbooks publication provides information about aspects of performing infrastructure health checks, such as checking the configuration and verifying the functionality of the common subsystems (nodes or servers, switch fabric, parallel file system, job management, problem areas, and so on).

This IBM Redbooks publication documents how to monitor the overall health check of the cluster infrastructure, to deliver technical computing clients cost-effective, highly scalable, and robust solutions.

This IBM Redbooks publication is targeted toward technical professionals (consultants, technical support staff, IT Architects, and IT Specialists) responsible for delivering cost-effective Technical Computing and IBM High Performance Computing (HPC) solutions to optimize business results, product development, and scientific discoveries. This book provides a broad understanding of a new architecture.

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, Clients, and IBM Business Partners from around the world create timely technical information based on realistic scenarios. Specific suggestions are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-8168-00 ISBN 073843924X