Redbooks
ibm.com/redbooks

IBM Spectrum Scale and IBM Elastic Storage System Network Guide

Kedar Karmarkar

John Lewars

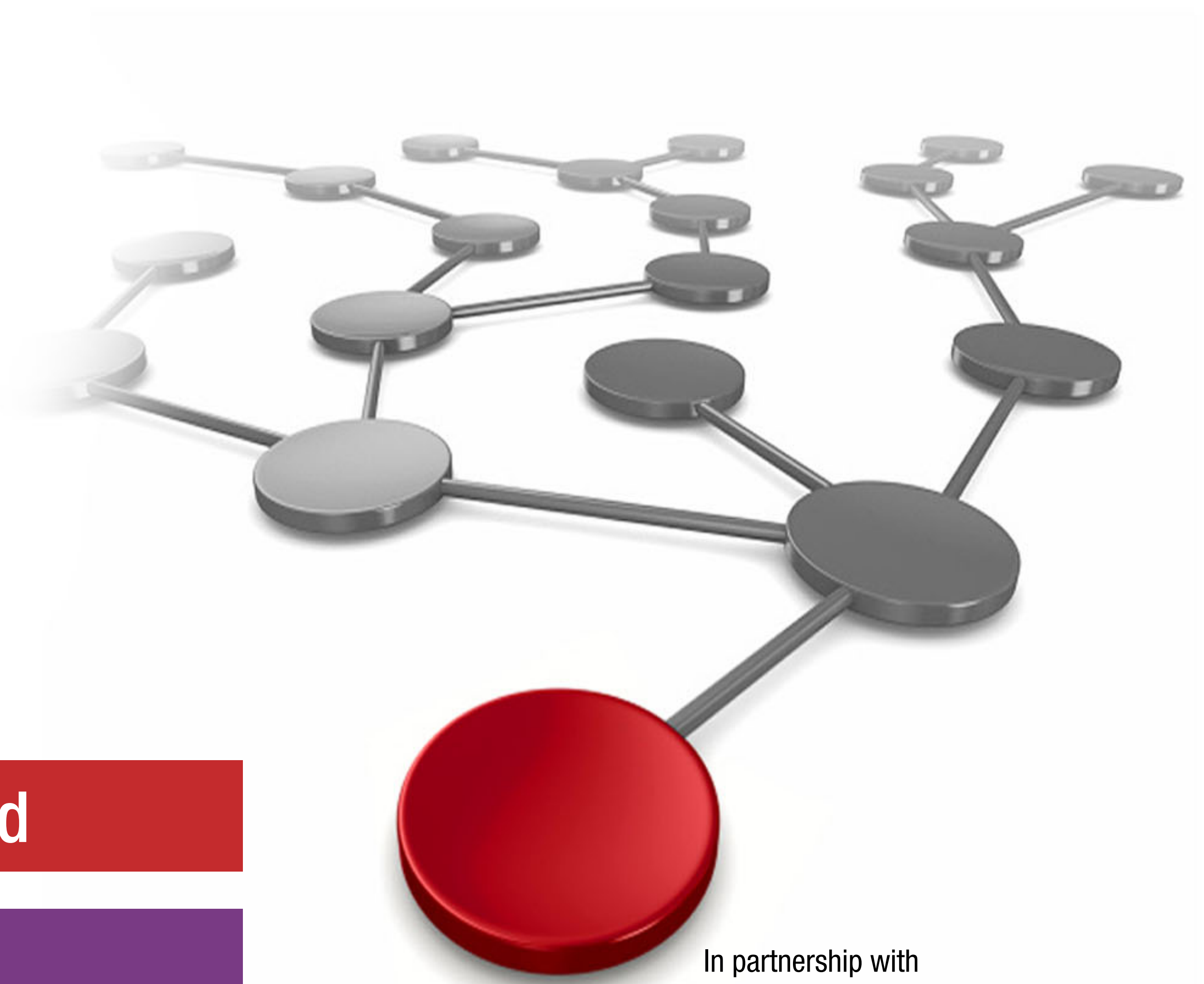
Sandeep R. Patil

Sandeep Naik

Kevin Gildea

Rakesh Chutke

Larry Coyne



Cloud

Storage

In partnership with
IBM **Academy of Technology**

IBM

Redpaper

International Technical Support Organization

IBM Spectrum Scale and IBM Elastic Storage System Network Guide

February 2021

REDP-5484-00

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

First Edition (February 2021)

This edition applies to Version 5, Release 1, Modification 0 of IBM Spectrum Scale and Version 5, Release 3, Modification 4 of IBM Elastic Storage Server.

This document was created or updated on February 17, 2021.

Contents

Notices

Trademarks

Preface

High-speed I/O workloads are moving away from the SAN to Ethernet and IBM® Spectrum Scale is pushing the network limits. The IBM Spectrum® Scale team discovered that many infrastructure Ethernet networks that were used for years to support various applications are not designed to provide a high-performance data path concurrently to many clients from many servers.

IBM Spectrum Scale is not the first product to use Ethernet for storage access. Technologies, such as Fibre Channel over Ethernet (FCoE), scale out NAS, and IP connected storage (iSCSI and others) use Ethernet though IBM Spectrum Scale as the leader in parallel I/O performance, which provides the best performance and value when used on a high-performance network. This IBM Redpaper publication is based on lessons that were learned in the field by deploying IBM Spectrum Scale on Ethernet and InfiniBand networks.

This IBM Redpaper® publication answers several questions, such as, “How can I prepare my network for high performance storage?”, “How do I know when I am ready?”, and “How can I tell what is wrong?” when deploying IBM Spectrum Scale and IBM Elastic Storage® Server (ESS).

This document can help IT architects get the design correct from the beginning of the process. It also can help the IBM Spectrum Scale administrator work effectively with the networking team to quickly resolve issues.

Authors

This paper was produced by a team of specialists from around the world working with Redbooks® Team, Tucson Arizona Center.

Kedar Karmarkar is a Senior Engineer and Solution Architect with the IBM Spectrum Scale development team. Kedar is part of the IBM Spectrum Scale Client adoption team and was the IBM Storwize® V7000 Unified Level 3 support lead in his earlier role at IBM. Kedar has over 20 years of infrastructure software, storage development experience in management, and architect roles. He has led development of network-attached storage (NAS), Block level virtualization, replication, systems, and storage management products. Kedar has a Bachelor of Engineering (Computer Science) degree from University of Pune, India.

John Lewars is a Senior Technical Staff Member with the IBM Spectrum Scale development team. He has been with IBM for over 20 years, starting first on the high-performance computing service team, working primarily on performance problems and code fixes, and then moving on to work in communication protocols and network management development. John has an extensive background in large parallel systems delivery and bring-up and led the technical computing performance development for years before moving to IBM Spectrum Scale team, where he has worked several assignments, including co-lead on Cloud team deliverables, such as support for AWS and container workloads. John's current assignment is IBM Spectrum Scale Performance Engineering Architect.

Sandeep R. Patil is a Senior Technical Staff Member who works as a Storage Architect with IBM System Labs. He has over 18 years of product architecture and design experience. Sandeep is an IBM Master Inventor, an IBM developerWorks® Master Author, and a member of the IBM Academy of Technology. Sandeep holds a Bachelor of Engineering (Computer Science) degree from the University of Pune, India.

Sandeep Naik is the Test Lead for IBM Spectrum RAID and IBM Elastic Storage System products. He has over 20 years of software development and test experience. He led the development of networking routers, SAN switches, SAN Storage, RAID technologies, and parallel file system. Sandeep holds a Bachelor of Engineering degree in Electronics and Instrumentation from SGSITS, Indore, India.

Kevin Gildea is a Distinguished Engineer with extensive experience in distributed computing, cluster computing, and high-performance interconnects. Currently, he contributes to the development of IBM Spectrum Scale. He holds a PhD in Computer Science from Rensselaer Polytechnic Institute and a BS in Computer Science from the University of Scranton.

Rakesh Chutke is IBM Storage and IBM Spectrum Scale consultant with IBM System Lab Services for the last 13 years. He has more than 19 years of IT industry experience working with clients across industries, including banking, insurance, telecommunications, media and entertainment, and oil and gas. Rakesh holds a bachelor of Engineering (Electrical engineering) from University of Mumbai, India.

Larry Coyne is a Project Leader fro IBM Redbooks®, Tucson Arizona Center. He has over 35 years of IBM experience, with 23 in IBM storage software management. He holds degrees in Software Engineering from the University of Texas at El Paso and Project Management from George Washington University. His areas of expertise include client relationship management, quality assurance, development management, and support management for IBM Storage Management Software.

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

Theodore (Ted) Hoover
Nariman Nasef
Kumaran Rajaram
IBM Systems

Special thanks to Scott Fadden, Tomer Perry, and Gowrishankar Muthukrishnan.

Now you can become a published author, too!

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

  `ibm.com/redbooks`

- Send your comments in an email to:

  `redbooks@us.ibm.com`

- Mail your comments to:

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

Stay connected to IBM Redbooks

- 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

IBM Spectrum Scale introduction

IBM Spectrum Scale is a proven, scalable, high-performance data and file management solution. It provides world-class storage management with extreme scalability, flash accelerated performance, automatic policy-based storage that features tiers of flash through disk to tape and support for various protocols, namely NFS, SMB, Object, HDFS, and iSCSI. The network plays a crucial role in successful deployments of IBM Spectrum Scale-based solutions. One of the key requirements for IBM Spectrum Scale deployment to excel stability is to have a reliable and stable underlying network.

This chapter includes the following topics:

- 1.1, “How is IBM Spectrum Scale different from other network storage technologies” on page 2
- 1.2, “Types of network traffic in IBM Spectrum Scale” on page 3
- 1.3, “High-level network flow for data and control traffic” on page 6

1.1 How is IBM Spectrum Scale different from other network storage technologies

IBM Spectrum Scale differs from other network storage technologies in two ways:

- Strict data consistency
- Parallel performance

Strict data consistency

IBM Spectrum Scale maintains POSIX lock compliance across the cluster to maintain data consistency. Therefore, IBM Spectrum Scale must communicate regularly between cluster members to maintain consistency. This process is optimized for efficient network utilization, although the lower the latency, the faster the consistency can be achieved. In comparison and for better understanding, NFS in most deployments is run with weak cache consistency that is designed more for a WAN-like topology. Therefore, IBM Spectrum Scale requires and relies on a stable network for efficiency and lower latency networks for better performance of many workloads.

Parallel performance

With other network storage technologies, such as NFS or iSCSI, access is typically from one consumer to one provider. An NFS client connects to a single NFS server for each export and an iSCSI target is typically associated with a single host. In contrast, each IBM Spectrum Scale node concurrently connects to all IBM Spectrum Scale I/O servers (also called a Network Shared Disk [NSD] server), distributing the I/O workload across all NSD servers; this parallelizing of I/O helps realize performance benefits.

This unique many clients-to-many I/O servers network traffic pattern is often a new workload on an established network infrastructure so the network might require some planning. The I/O flow from IBM Spectrum Scale Client to IBM Spectrum Scale NSD servers is shown in Figure 1-1.

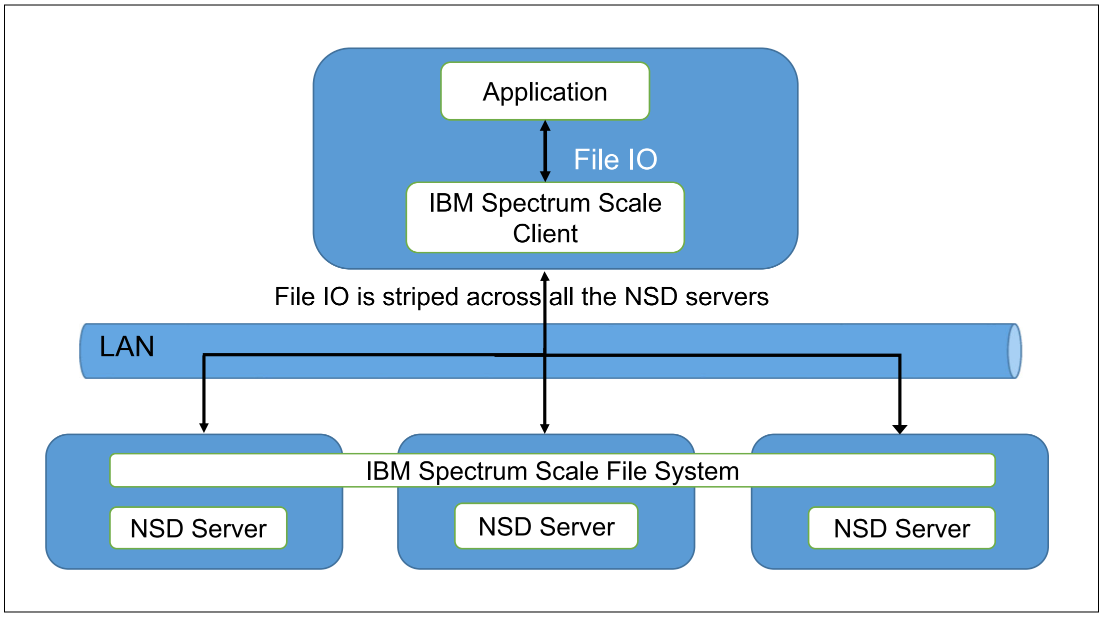

Application
File IO
IBM Spectrum Scale Client
File IO is striped across all the NSD servers
LAN
IBM Spectrum Scale File System
NSD Server
NSD Server
NSD Server

Figure 1-1 Striping of File I/O across IBM Spectrum Scale NAS servers

1.2 Types of network traffic in IBM Spectrum Scale

Network traffic in IBM Spectrum Scale can be largely attributed by the factors:

- Low latency versus high throughput network for performance
- Communication between nodes in cluster (internally or externally)
- Features that are added to cluster configuration, such as Active File Management (AFM), Encryption, or providing access through NFS or SMB or object

This section describes the types of network traffic.

Note: File system operations exist that are sensitive to network latency and others are sensitive to bandwidth. Which aspect of network behavior is important to you is based on your I/O workload. For example, opening a file is sensitive to latency because at minimum a network message (RPC) is available to acquire a token and a message to obtain the inode. Whether this is important to your application can depend on the size of the files. If the read or write operation performs large sequential I/O requests against a large file, file open time might not be noticeable. However, if starting your application requires opening 10,000 odd files, having a low latency network is critical.

IBM Spectrum Scale network traffic can be broken down into the following types of traffic:

Control Traffic	In-Band[1] and Latency dependent; internal file system and protocol communication that is related to IBM Spectrum Scale (such as a token request, which is used to determine the right of node to perform file operation in a specific cluster, space allocation, file system quota update, SMB lock traffic between protocol nodes, managing metadata updates from nodes acting as metanode, and so on).
Data Traffic	In-Band; specific to file data including metadata (inode) and data and is sensitive to latency or bandwidth.
Administration	Out-of-Band[2] where operation not highly dependent on low latency or high bandwidth.

For more information about IBM Spectrum Scale Token, metadata, and CES, see these resources:

- IBM Knowledge Center
- IBM Spectrum Scale

Control traffic of the core file system is one of the key traffic types for the wellness of the cluster, which is described next.

[1] In this context, In-Band represents operations in the data access path, where an increase in latency or a decrease in bandwidth is noticed by an application reading or writing data.

[2] In this context, Out-of-Band refers to asynchronous operations that generally are not sensitive to high latency or low bandwidth until these attributes hit an extreme number of concurrent executions.

1.2.1 Control traffic

Control traffic in IBM Spectrum Scale environments can be divided into the following major types:

- Cluster “well-being”; that is, disk leases (“heartbeat”)
- Data access or consistency management (tokens, quotas, allocation, and so on)
- Configuration or feature related (key management, Zimon collectors, configuration management, AFM-related RPCs, and so on)

Most of these types of communication are latency sensitive (that is, slow response time affects performance). However, some of these types, especially “well-being”, are less latency sensitive, but greatly affect the availability of nodes in the cluster if message delay exceeds specific values (for example, heartbeat timeouts).

To understand the different control traffic topic, it is important to understand the special management functions that nodes can play in a cluster. Therefore, we recommend that you see this page of IBM Knowledge Center.

Disk leases control traffic

IBM Spectrum Scale uses a mechanism that is referred to as a *disk lease* to prevent file system data corruption by a failing node. A disk lease grants a node the right to submit I/O to a file system. File system disk leases are managed by the cluster manager of the file system’s home cluster. A node must periodically renew its disk lease with the cluster manager to maintain its right to submit I/O to the file system.

Disk lease is an important type of control traffic. The cluster manager is automatically elected from the list of “quorum nodes”. Each node in a cluster (regardless of its role) must renew its lease with the cluster manager server at regular intervals (35 seconds by default).

Disk lease is a key control traffic to maintain the well-being of the cluster network traffic. When a node fails to renew a disk lease with the cluster manager, the cluster manager marks the node as failed, revokes the node’s right to submit I/O to the file system, expels the node from the cluster, and starts the recovery processing for the failed node.

Token traffic

IBM Spectrum Scale uses what is called a *token* to keep track of what nodes have the right to create, modify, or read a file. IBM Spectrum Scale uses these tokens to implement POSIX locking and to determine which server or servers that are running IBM Spectrum Scale software has the lock and which must wait. That is, token traffic is a type of data access or consistency management control traffic.

As a distributed file system, IBM Spectrum Scale needs some mechanisms to assure data consistency and capacity management. One of the basic constructs of IBM Spectrum Scale as a parallel file system is the use of tokens to maintain consistency. Essentially, to perform an operation on a file system object (file, directory, and so on), a relevant token is required for reading, writing, caching, and so on. At least one token server is automatically selected from the list of “manager” nodes. The token servers manage the tokens, and each client (regardless of its role) must contact a token server.

Distributing the tokens across multiple token manager nodes allows more tokens to be managed or retained concurrently, which improves performance in situations where many lockable objects are accessed concurrently. Token traffic is based on small RPC messages, which are mostly latency sensitive (that is, not throughput oriented).

Metanode traffic

Control traffic is not limited to communication between NSD clients and manager nodes because it occurs regularly peer-to-peer and can include any nodes in the cluster. When a file is opened on a node, the node becomes the metadata manager for that file that is performing metadata updates and coordinating access to that file from multiple nodes. When a node owns the metadata for a file it is referred to as the metanode. Metanode traffic is similar to generic token traffic, that is, latency-sensitive for small messages. Metanode traffic is a type of data access/consistency management control traffic.

Other control traffic

The following other types of control traffic are available:

- When IBM Spectrum Scale is configured for file encryption with an external Key manager, key fetching from key servers is mostly around small messages to the designated set of servers.
- IBM Spectrum Scale performance collection modules (Zimon) collect various performance-related metrics. Zimon reporting is also based on relatively small messages that flow between monitored nodes to their designated collector.
- Cluster Configuration Repository (CCR)-related communication is used for updating and querying configuration-related information. All quorum nodes act as a CCR node, meaning that they manage the distributed configuration state of the cluster. CCR communication often uses small messages from all cluster nodes. Special nodes that use the CCR more than others (Protocol nodes, GNR nodes, GUI, and so on) might introduce more traffic than others.
- When IBM Spectrum Scale is configured for AFM, control traffic between the AFM gateway node and clients occurs in the form of special RPC to notify on the file system operations that must be replicated by the gateway. The AFM-related RPCs are small messages, because only the file system operations details are being transmitted, not the data. The traffic might flow to or from any node that access the AFM file set (on local or remote clusters).

Note: This section covers only the control traffic of IBM Spectrum Scale clustered file system and does not cover the control traffic that is related to supported protocols by IBM Spectrum Scale (NFS, SMB, Object, and HDFS). The control traffic IBM Spectrum Scale file system by design has relatively stringent requirements on the network for performance reasons, unlike the supported protocols that are relatively more tolerant by design. Therefore, it is vital to cover control traffic of IBM Spectrum Scale clustered file system in the context of this paper.

1.2.2 Data traffic

For simplicity, we divide the data traffic into two logical halves:

- File system-related data traffic: This traffic is the data transfer between the IBM Spectrum Scale NSD servers and various IBM Spectrum Scale clients (which might have other roles, such as protocol nodes, IBM Spectrum Protect nodes, AFM GW, and TCT nodes). In this context, data also includes file system metadata. This data traffic uses NSD protocol for communication.
- Protocol-related data traffic: Protocol nodes (NFS, SMB, Object, and HDFS) uses file system-related data communication at the backend to communicate with the NAS servers. On the front end, protocol nodes use the relevant data path to communicate with its clients.

1.2.3 Administration traffic

Administration traffic relates to administrative operations that occur across the cluster by way of remote shell commands, such as `ssh` and in some cases by way of IBM Spectrum Scale RPC. It also includes the GUI-based administration and REST API-based administration. The performance requirements of this type are not critical.

1.3 High-level network flow for data and control traffic

Figure 1-2 shows the control and network traffic in a IBM Spectrum Scale cluster.

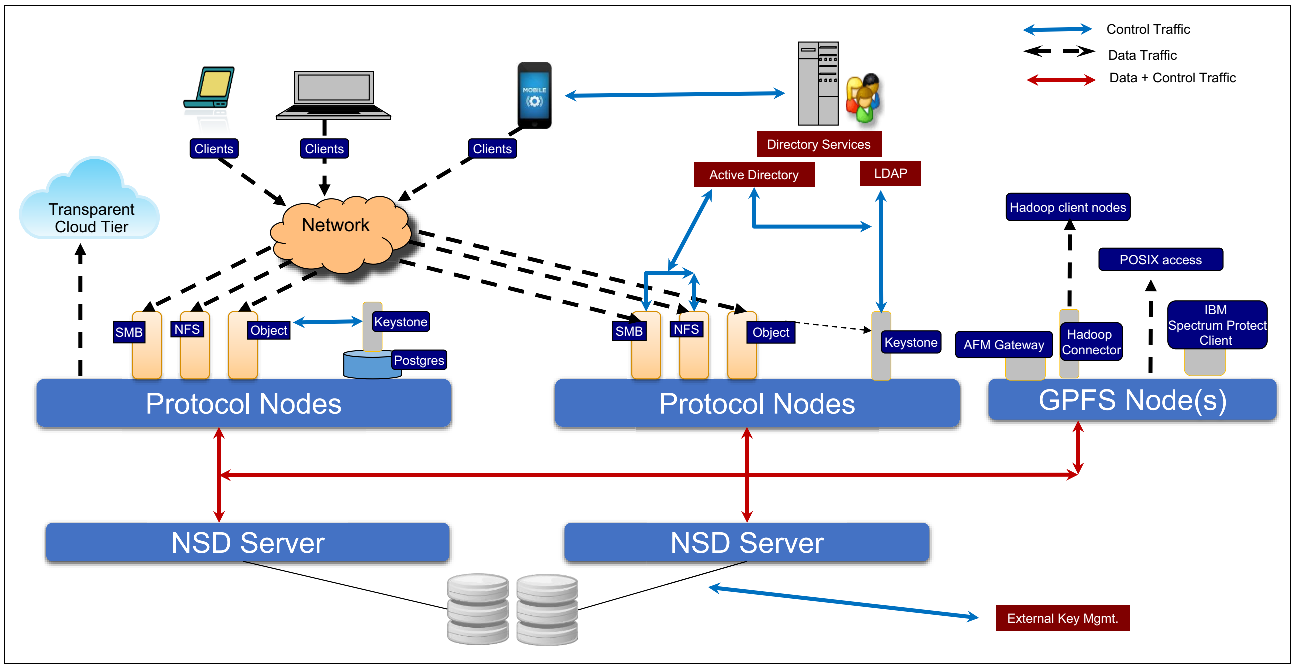


Figure 1-2 IBM Spectrum Scale network flow for data and control traffic

1.3.1 IBM Spectrum Scale supported networks

IBM Spectrum Scale solutions can be deployed with the following networking configurations for its inter-cluster and protocol communication:

- 1-Gigabit Ethernet

  1 Gbps Ethernet is a common standard that is used in networking environments. Although IBM Spectrum Scale supports this standard, we recommend the use of high-bandwidth network for data and cluster traffic.

- 10-Gigabit Ethernet

  10 Gbps Ethernet is a technology that is widely adopted in many data centers. This option is useful for clustered environments with increased communication demands. Even if the bandwidth might not be a concern in your environment, teaming configurations should be considered for high availability of the IBM Spectrum Scale inter-node communication network. 10 Gbps Ethernet and higher bandwidth adapters are proven to be efficient in most environments.

- 40 and 100-Gigabit Ethernet

  The 40 Gbps and 100 Gbps Ethernet technology is gaining traction and is another option for enhancing the IBM Spectrum Scale communication for environments with larger bandwidth demands. Now IBM Spectrum Scale can use the underlying 40 Gbps and 100 Gbps Ethernet technology with TCP/IP communication protocol on all IBM Spectrum Scale supported platforms: AIX®, Linux, and Windows operating system.

- InfiniBand networks

  IBM Spectrum Scale can also use IP over InfiniBand and Remote Direct Memory Access (RDMA) fabric to provide access to the file system. It is recommended to use InfiniBand networks for IBM Spectrum Scale cluster communication when high performance is a critical aspect of the deployment.

- RDMA over Converged Enhanced Ethernet

  Many times, Ethernet is often the choice in data centers; therefore, it is essential to bring the advantages RDMA provides to IBM Spectrum Scale into the Ethernet environment as well. IBM Spectrum Scale supports RDMA over Ethernet networks by encapsulating InfiniBand style headers in Ethernet packets. This feature extends the IBM Spectrum Scale support for the use of RDMA over InfiniBand to RoCEE environments.

Note: RDMA over Ethernet (RoCEE) is not supported on IBM Enterprise Storage Server.

An IBM Spectrum Scale deployment with mixed networks can be used, such as having an InifiniBand network for traffic between servers while having Ethernet for communication with the clients. For more information about planning the network for IBM Spectrum Scale deployment, see Chapter 2, “Network planning and best practices for IBM Spectrum Scale and IBM Elastic Storage System” on page 9.

1.3.2 IBM Spectrum Scale expels

In the context of this publication, it is important to understand expels in a IBM Spectrum Scale cluster. As the name indicates, expels in an IBM Spectrum Scale cluster is a scenario in which a specific node that is a part of the cluster is expelled from the cluster. Many scenarios exist where expel of node or nodes occurs that might directly affect the workload; therefore, it is important to understand the reason for the expel. Field experience shows that many expel scenarios relate to the underlying network. Consider the following important expel cases that are mostly associated with the control traffic:

- If a node fails to renew its lease, the cluster manager attempts to ping the “suspected node” before expelling it. For example, in some cases, the IBM Spectrum Scale lease renewal message is delayed because of excessive packet drops and the subsequent TCP recovery. In such a case, the delay can exceed the failure detection time and the node is expelled from the cluster. This type of an event can occur even while other data operations appear to be proceeding.

- IBM Spectrum Scale uses a mechanism that is referred to as a *node expel request* to prevent file system resource deadlocks. Nodes in the cluster require reliable communication among themselves to coordinate sharing of file system resources.

  If a node fails while owning a file system resource, a deadlock can ensue. If a node in the cluster detects that another node that owns a shared file system resource maybe failed, the node sends a message to the file system cluster manager requesting the failed node to be expelled from the cluster to prevent a shared file system resource deadlock. When the cluster manager receives a node expel request, it determines which of the two nodes should be expelled from the cluster and takes similar action as described for the disk lease expiration. A node expel request might occur if a node cannot communicate with another node, which can be because of network issues.
- If a metanode is experiencing a network issue that causes high latency, it can affect other nodes in the cluster. This result is why you might see one NSD client ask for another NSD client to be expelled. It is asking because it needs something from that other client and the client is not responding in a reasonable period.
- Token traffic is based on small RPC messages, which are mostly latency sensitive (that is, not throughput oriented). Slow response time (because of latency, network congestion, and so on) might result in “hangs” or “hiccups” from a user or application perspective.
- File system allocation and quota-based control traffic is between the nodes and cluster manager that concern block/inode or quota allocation. Block/inode allocations are based on small messages and therefore are latency sensitive. Although regular allocation might not be affected by slowness because of the IBM Spectrum Scale region-based approach of allocation, quotas allocation might affect write or create performance.
- When a node fails to renew a disk lease with the cluster manager, the cluster manager marks the node as failed, revokes the node’s right to submit I/O to the file system, expels the node from the cluster, and starts recovery processing for the failed node.

2

Network planning and best practices for IBM Spectrum Scale and IBM Elastic Storage System

IBM Spectrum Scale is a software-defined storage product for high-performance, large-scale workloads. IBM Spectrum Scale (formerly IBM General Parallel File System or GPFS) is a scalable data and file management solution that provides a global namespace for large data sets along with several enterprise features. IBM Spectrum Scale is used in clustered environments and provides file protocol (POSIX, NFS, and SMB) and object protocol (Swift and S3) access methods.

IBM Elastic Storage System is a software-defined storage system that is built upon proven IBM Power® Systems, IBM Spectrum Scale software, and storage enclosures. IBM Elastic Storage System allows scale up for capacity or scale out for performance in modular building blocks, which enables sharing for large data sets across workloads with unified storage pool for file, object, and Hadoop workloads. IBM Elastic Storage System uses IBM-developed erasure coding-based declustered RAID technology to rebuild failed disks in few minutes.

This chapter includes the following topics:

2.1 Network planning and sizing

This section describes various considerations for designing and selecting the IBM Spectrum Scale and IBM Elastic Storage System environment. Various factors that decide the size, speed, resiliency, and redundancy of network that can be used for IBM Spectrum Scale communication also are described.

The Internet Protocol network is the most foundational element in building overall IBM Spectrum Scale solution. In IBM Spectrum Scale, cluster compute nodes are tightly coupled with each other by using IP network to continuously exchange the state of each member in the cluster and therefore represent as single entity or namespace to applications workload.

2.2 Bandwidth requirements

In this section, we describe the requirements for aggregate cluster and single node bandwidths.

2.2.1 Aggregate cluster bandwidth

Aggregate bandwidth is the use of the bandwidth when more than one node in the cluster is active at a time, catering application processing, or performing read/write to the same file system concurrently.

Aggregate bandwidth should be estimated correctly for parallel processing applications, such as high-performance computing (HPC) and cluster-aware applications where tasks are distributed across multiple nodes by using some task scheduler or by applications.

You might have to ensure that all network shared disk (NSD) servers are equipped with sufficient infrastructure to support the aggregate bandwidth that is demanded by the clients.

NSD servers should support enough host bus adapters (HBAs) to handle required aggregate bandwidth from back-end storage and enough network adapters to dispatch it quickly to front-end clients.

The overall storage, network, and NSD server sizing can be done based on aggregate bandwidth demand of the compute clients. When you calculate aggregate bandwidth, you might have to keep extra headroom for future growth of application workload.

The aggregate bandwidth that is needed at SAN level or at network level depends on the IBM Spectrum Scale topology that is used (see Figure 2-1).

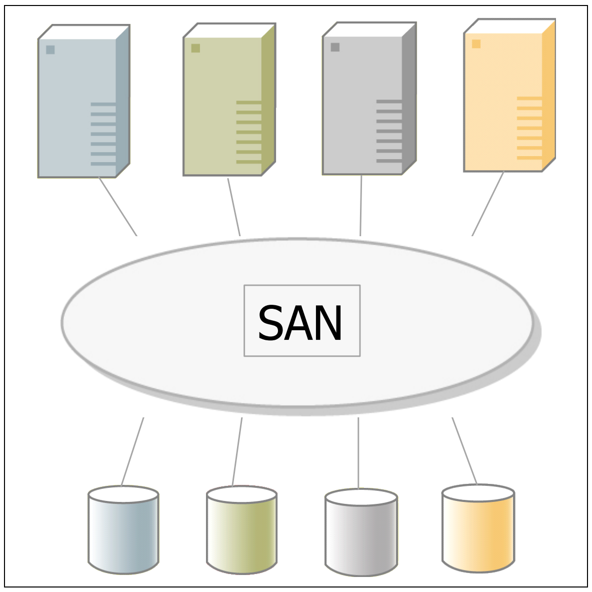


Figure 2-1   IBM Spectrum Scale shared disks (SAN-attached)

A SAN-attached model in which all nodes are directly attached through SAN and sharing LUNs and file systems is shown in Figure 2-1. Applications directly run on each of these nodes to achieve highest performance. In this case, the aggregate node bandwidth points to SAN bandwidth, which should be designed based on what maximum storage can deliver.

Although IP networking is still essential part of the IBM Spectrum Scale cluster, it is used only for administration traffic; therefore, a high-speed network is not needed for a SAN-attached model.

An NSD model includes NSD servers to drive IOs for NSD clients (see Figure 2-2). It is assumed that no applications are active on NSD servers and they are intended to serve all applications.

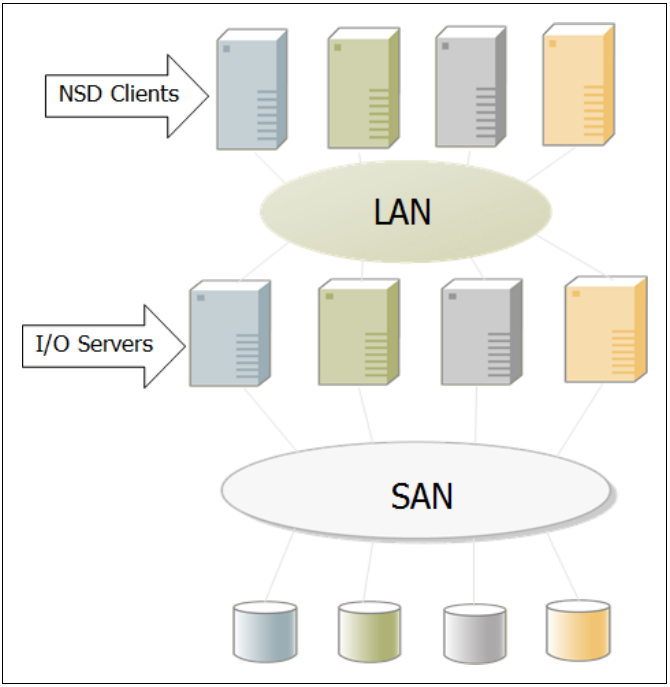


Figure 2-2 IBM Spectrum Scale Network block IO-NSD model

All applications that are running on NSD clients (and therefore, the designing aggregate bandwidth from client point of view) is important in this case. Two elements from the client point of view are to be considered: aggregate network bandwidth and aggregate SAN bandwidth.

In this case, aggregate clients network bandwidth should be equal to individual NSD server SAN bandwidth because multiple clients might be reading and writing to same NSD disk (which is associated with only one NSD server) at time.

Access to the IBM Spectrum Scale file system data by using SMB, NFS, and object protocol is shown in Figure 2-3 on page 13. These four NSD servers are also acting as protocol nodes for top clients. They access IBM Spectrum Scale file system data by using one of the protocols.

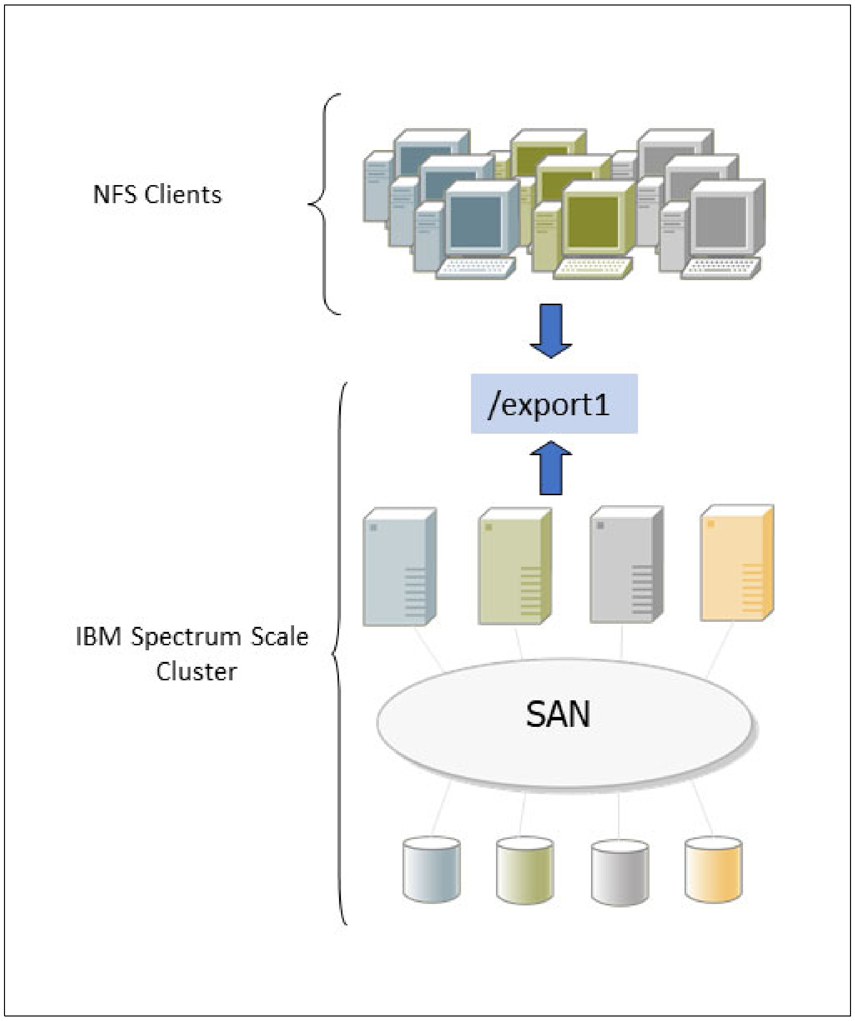


Figure 2-3 IBM Spectrum Scale for CES protocols

We assume that each of the protocol node can cater any protocol, including NFS, SMB, and Object to the end clients.

Here, the total number of clients that are divided by number of protocol nodes shows how many client connections each protocol node can handle. Therefore, the aggregate bandwidth that is needed for each protocol node depends upon total number of client nodes that are communicating with protocol node at a time.

For example, if four protocol nodes and 40 clients exist, each protocol node takes up maximum 10 client connections (considering DNS-based load balancing is deployed, if not consider that the administrator manually performed equal distribution of the number of clients across each protocol node).

Assuming that all of these 10 clients are active at a time, they need a specific amount of aggregate bandwidth (that is, the aggregate client bandwidth that is needed to drive workloads from those 10 clients at a time). This aggregate client bandwidth value must be the same or more that each protocol node can support, which further translates to protocol node client communication network bandwidth and SAN bandwidth to communicate with back-end storage (assuming that the underlaying storage can deliver the required bandwidth).

It is good idea to establish actual bandwidth usage during peak use periods for accurate estimation.

2.2.2 Single node bandwidth

Estimating single node bandwidth helps you to correctly size nodes for CPU, memory, the number of HBA (FC ports), and number of network adapters and their type.

After you know the single node bandwidth requirement, you can use the correct quantity and specification of hardware that can deliver the required single node bandwidth. The sizing of an NSD server required proper consideration while selecting number of HBA (FC cards) and network cards because they drive high traffic to storage and clients.

Clients that are accessing the IBM Spectrum Scale file system over TCP/IP must have the adequate number and speed of the network adapter ports. Generally, it can be estimated based on the application throughput requirement.

Ensure that the applications (which run on the client nodes) leave some CPU cycles and network bandwidth for IBM Spectrum Scale daemon to function. The CPU-intensive applications that attempt to use every bit of CPU might not leave enough CPU cycles for IBM Spectrum Scale daemon to function, which can result in a node expelled issue. Similarly, a busy network might block IBM Spectrum Scale communication, which results in a node expelled-related issue.

2.3 Deciding network bandwidth planning

The following factors must be considered as you plan for network bandwidth:

- Synchronous replication
- Asynchronous replication
- Network latency
- Access pattern
- Type of data set
- Ability of back-end storage to support required bandwidth
- Future growth

Synchronous replication

Synchronous replication requires each write to be completed successfully on the primary and auxiliary storage failure group disks before the servers or initiators can send the next set of write data.

To avoid any response time issues, determine the peak sustained write rate for your server and ensure that adequate bandwidth is available to manage this load and to allow for future growth. Only writes are replicated to the secondary failure group disks.

For a typical system, this peak is usually during batch processing, when many updates are occurring. It might be higher still during month-end or year-end processing. You must make allowances for this peak period when you perform your bandwidth calculations.

Data replication, particularly in synchronous mode, requires in-order packet delivery, latency tolerances in the handful of millisecond range, and network path redundancy.

Deterministic optical networks can provide all of these requirements most effectively. An optical network can deliver data 200 kilometers (124.27 miles) in 1 or 2 milliseconds.

Asynchronous replication

Unlike synchronous replication, the asynchronous replication process does not hold application writes to be completed on a secondary file set.

The primary site is a read-writable file set in which the applications are running and they have read/write access to the data. The secondary site file set is recommended to be read-only. All data from the primary site is asynchronously replicated to the secondary site.

A consistent point-in-time view of the data in the primary file set can be propagated inline to the secondary file set with the use of fileset-based snapshots (psnaps). The recovery point objective (RPO) feature makes it possible to specify the frequency of these snapshots.

Because of the time lag, data on the remote systems is not always a mirror of the data that is at the primary site.

To determine the bandwidth requirement for your asynchronous replication, consider the amount of data you must transfer to the remote site and the available RPO window.

Larger is the RPO; lesser is the bandwidth that is required for replication.

Network latency

Providing the network with sufficient bandwidth is a necessary (but not sufficient) step to ensuring good performance of a networking application. If excessive network latency is causing the application to spend a large amount of time waiting for responses from a distant data center, the bandwidth might not be fully used and performance suffers.

Network latency is the time that it takes for a packet to travel across the network. It is usually measured and reported as the Round Trip Time (RTT). An easy way to measure RTT is by using the `ping` command. The `traceroute` command goes a step further and reports the RTT for each hop between the local system and the remote system.

In most situations, the dominant factor that determines latency is the physical distance that packets must travel. Not much can be done about the actual distance between the two systems that communicate, but sometimes packets are routed over longer paths than necessary.

Network I/O latency also depends on how fast the target storage can process the data or the number of systems that are injecting data to storage elements. If more systems are pumping data simultaneously to single storage system, it might so happen that storage might not be quick enough to process the aggregated data within an accepted response time. This issue might further delay I/O and higher response times can be seen for completing the I/Os.

Access pattern

IBM Spectrum Scale attempts to recognize the pattern of accesses (such as strode sequential access) that an application makes to an open file. If IBM Spectrum Scale recognizes the access pattern, it optimizes its own behavior. For example, it can recognize sequential reads and retrieve file blocks before they are required by the application.

When describing volumes of data, things are slightly different. Bandwidth is usually used to describe the maximum theoretical limit of data transfer, while throughput is used to describe a real-world measurement. It might be said that bandwidth is the maximum throughput. Bandwidth and throughput figures are often given in units of size over units of time; for example, Mbps or GBps.

IOPS and throughput were related by the relationship that is shown in Figure 2-4.

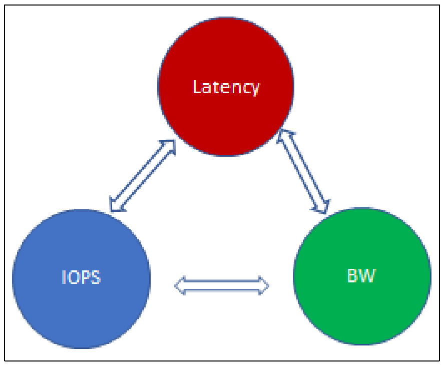


Figure 2-4 Throughput = IOPS x I/O size

Now is the time to start thinking about that I/O size. If we read or write a single *random* block in 1 second, the number of IOPS is 1 and the I/O size is also 1 (a unit of "blocks" is used here to simplify this example). The throughput can be calculated as (1 x 1) = 1 block/second.

Alternatively, if we wanted to read or write eight contiguous blocks from disk as a *sequential* operation, it again results only in the number of IOPS being 1. However, the I/O size is 8 this time. The throughput is calculated as (1 x 8) = 8 blocks/second.

This example shows the great benefit of sequential I/O on disk systems because it allows increased throughput. Whenever you increase the I/O size, you get a corresponding increase in throughput, while the IOPS figure remains resolutely fixed. But, what happens if you increase the number of IOPS?

When a single-threaded process is reading or writing a single random block on a disk, the I/O results in a certain amount of latency. This result occurs because of the seek time and rotational latency of the disk.

We know that the average rotational latency of a 15 k RPM disk is 4 ms, so we add another millisecond for the disk head seek time in our example and call the average I/O latency 5 ms. How many (single-threaded) random IOPS can we perform if each operation incurs an average of 5 ms wait? The answer is 1 second/5 ms = 200 IOPS. Our process is reaching a physical limit of 200 IOPS on this disk.

What do you do if you need more IOPS? With a disk system, you have only one choice: add disks. If each spindle can drive 200 IOPS and you require 80,000 IOPS, you need (80,000/200) = 400 spindles.

However, if you can perform the I/O sequentially, you might be able to reduce the IOPS requirement and increase the throughput, which allows the disk system to deliver more data.

Data set

Two types of data are available: small-size file and large-size file. Medium-size files might exist, but we are considering only small and large-size files for simplicity.

Now, we know when too many small files are generated by an application, such workloads generate huge metadata. It is important in these specific cases that you might move metadata on faster disks, such as SSD or Flash storage to improve file system performance.

Usually, workloads that generate large-size files are not metadata-intensive and therefore data and metadata can coexist on the same NSD without affecting performance.

Large-size file processing indicates sequential behavior of access patterns in which throughput is needed for processing them faster. For such workloads, consideration must be given to network bandwidth and IBM Spectrum Scale file system caching parameters.

Applications that are processing smaller files show the random nature of workloads that generally benefit for multithreaded applications and faster storage disks.

Ability of back-end storage to support required bandwidth

After you establish the aggregate bandwidth that is required for the application to process data from multiple workstations in concurrent fashion, you can then start planning other dependent elements in the solution. Consider adding at least 20% more than the required bandwidth for future growth.

A general rule says that your network bandwidth should be at least 30 - 35% greater than back-end storage subsystem bandwidth or you can even plan more by predicting storage growth ahead of time.

Future growth

You might need more network bandwidth if your back-end storage supports scalability where it allows adding disk enclosures to its controllers. With disks added to your storage system now, more throughput can be driven so that bottlenecks might shift to host adapters (FC or NIC). This shift limits the host's ability to take advantage of increased storage bandwidth. Therefore, it is imperative to plan proper network bandwidth and consider future growth of peripheral elements in the solution.

It can be good idea to understand about the maximum bandwidth that storage subsystem can support, considering its full blown capability during initial network planning phase.

Also, consider how much storage will be added later to support the solution and plan the network infrastructure accordingly.

2.4 Network design considerations

An inter-datacenter storage network infrastructure to support data replication generally includes the following characteristics:

- Low latency
- Minimal packet loss
- Flow control mechanisms
- Network resiliency
- Scalable bandwidth

Data replication, particularly in synchronous mode, requires in-order packet delivery, latency tolerances in the handful of millisecond (one-thousandth of a second) range, data rates per communication link near 1 G or 2 Gbps, and network path redundancy. Deterministic optical networks can provide all of these parameters most effectively.

An optical network can deliver data 200 kilometers (124.27 miles) in 1 millisecond, or from New York to Los Angeles in 20 milliseconds. Optical networks also are scalable. Restoration times if a fiber is cut on a transport path are under 50 milliseconds.

Unlike an IP-based network, Sonet/SDH and C/DWDM are designed for loss-less data transmission, which is an important factor in a data replication environment. Optical networking equipment now provides buffer credits and caching mechanisms for flow control in SAN environments to enable efficient use of the network link.

Resiliency and availability

Resiliency in the context of a resilient network is the ability of the network, a device on the network, or path on the network to respond to failure, resist failure, handle flux in demand, and easily shift and configure with little or no effect on service delivery.

A resilient network is the agent that can help diminish the loss of employee productivity if a major disaster occurs. The major elements of resilient network design are shown in Figure 2-5.

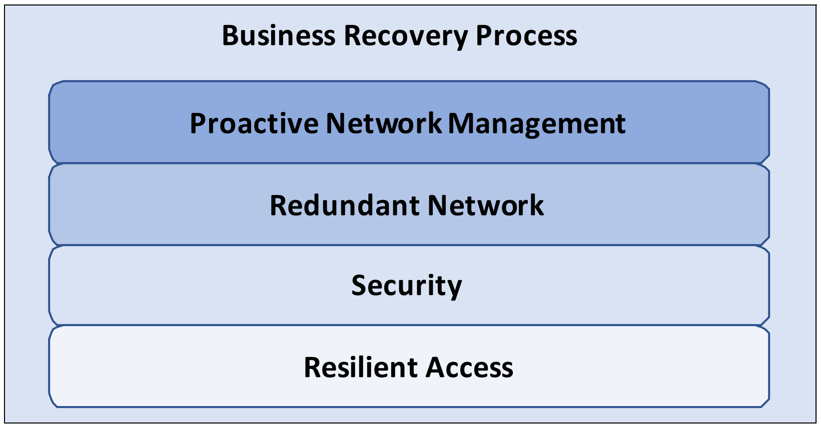


Figure 2-5 Resilient network design

Because businesses are becoming increasingly dependent on intra- and inter-organizational communications, the resilient, redundant network provides a key element to the resilient enterprise.

For the resilient access mechanisms or processes to gain access, they must first qualify access through a security process. If the security process is satisfied, users gain access to a resilient communications network. That network then allows users to connect and interact with the suitable business processes within the resilient enterprise.

The scope of a resilient network infrastructure can be determined by considering the number of the following parameters:

- Paths
- Ports
- Switches
- ISL

Number of paths

How the network, device, or path reacts to failure is determined in advance so that predictable network, device, or path conditions are present after response to failure. Here, the number of paths that is required to reach to target devices is configured by considering the availability and redundancy requirements for the application. More paths that target the device not only services the adapter or port failure, but also provides more bandwidth and thus, better performance.

Number of ports

The number of network ports and the speed should be selected based the speed of switching element and throughput requirement of single host.

Having more ports gives better options for an operating system user to implement link aggregation techniques, such as LACP, bonding, and teaming. The link aggregation allows applications to pump more traffic through multiple ports at the same time, which directly benefits the application to perform faster read and write to the device.

Number of switches

Resilient networks incorporate device connectivity through many switching elements to form highly available network. The resilient network architecture includes redundant (multiple) network switches that can take over the function of one another if one fails.

The link aggregation allows single IP to communicate through two different network switches, which provides switch level redundancy.

Number of ISL

The number of ISL should be good enough to cater required bandwidth to the user or device without compromising the quality of service.

Size and future growth

No straightforward formula is available to size NSD servers for use with IBM Spectrum Scale. An NSD server does not use much memory or CPU. The key is to ensure that it can support the required I/O throughput. From a best practices perspective, it is suggested to avoid the use of NSD server for porting more functions, such as gateway node for AFM and IBM Spectrum Archive node.

The required PCI bandwidth can be determined by how much performance is needed. In IBM Spectrum Scale, this process is called designing the storage “building block.” A storage building block is a pairing of Storage devices and NSD servers that provide a specific capacity or level of IO performance. It usually includes two or more NSD servers and some storage.

Two or more NSD servers provide availability because performance usually depends on the available network connections. The stack to consider includes the storage, server hardware, and network connection to the rest of the cluster (see Figure 2-6).

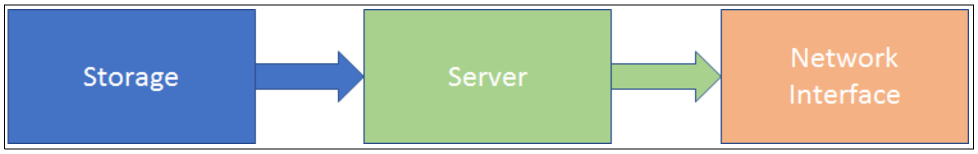


Figure 2-6 Connections stack

Designing the performance for a building block means matching the performance of your storage with the network. For example, your storage controller provides 2 GBps of sequential throughput, and you have two NSD servers and each NSD server needs one 10 Gb link to provide full throughput that might be minimal requirement. However, from a planning or best practices perspective, it is advisable to design the NSD server with enough network adapters so that each adapter can use the entire bandwidth of back-end storage. If one of the NSD servers fails, the available server provides close to full storage bandwidth to the compute clients.

If the storage is faster, you might need more 10 Gb networks or NSD servers, or you might need to use a different network type, such as 40 Gb Ethernet, InfiniBand, or Omni-Path. To get the best performance out of your storage hardware, ensure that the NSD servers can provide full throughput of the storage all the way to the clients (through the network).

Assessment of network infrastructure

When you plan for the network on which the IBM Spectrum Scale cluster is to be installed, ensure that you closely review the network infrastructure. You might need to ask the customer to share more information about the following high-speed network management network connectivity details:

- What network topology is used?
- Does the switching element support an LACP configuration?
- What is the hash mode that is supported by network switches for balancing load when LACP is configured?
- Are sufficient free ports available for connecting IBM Spectrum Scale I/O servers?
- What is the speed of the network port that is supported by switch? Does that match with I/O server network adapter speed?
- Does network switch support quality of service?
- Is Tx and Rx flow control enabled on the network switch?
- Is the customer running on latest level of switch operating system?
- What is the number of hops to reach to clients from I/O server?
- Does the switch support Jumbo frame?
- Does the customer have NTP?
- Does the customer have DNS?

2.5 OSI Layer

The Open Systems Interconnection (OSI) model is a conceptual model that is commonly used to characterize and standardize the networking system. We use this model to explain important concept and consideration at each layer in the context of IBM Spectrum Scale.

2.5.1 OSI Layer 1: Physical layer

The physical layer, which can include HBAs and cables, defines the means of transmitting raw bits from one node to another node. In this publication, we focus on only the most important aspects of the physical layer.

PCI bus

It is good idea to specify the PCI bus speed table for the benefits of readers (HBAs are generally plugged into PCI Express slot in server). PCI Express has versions 1 - 4 and Physical PCI Express links can contain 1, 2, 4, 8, 12, 16, or 32 lanes.

The final throughput that you can get from the HBA depends on the version and number of lanes. At the higher speed, the PCI bandwidth can be the limiting factor. For example, two-port 100 Gbps HBAs cannot offer you total 200 Gbps throughput because of current PCIe limitations.

Memory bandwidth and NUMA

Most modern CPUs support more than one system bus, each serving a small set of processors. Each group of processors has its own memory and I/O channels. This group is called a Non-uniform memory access (NUMA) node. Under NUMA, a NUMA node can access its own local memory faster than non-local memory (that is, memory of other NUMA node), hence the NUMA name. The number of CPUs within a NUMA node depends on the specific hardware vendor.

Intel and IBM POWER® processors include NUMA. Because NUMA uses local and foreign memory, it takes longer to access some regions of memory than others. Local memory and foreign memory are typically used regarding a running thread.

Local memory is the memory that is on the same node as the CPU that is running the thread. Any memory that does not belong to the node on which the thread is running is foreign.

When IBM Spectrum Scale software is running on NUMA-based system, where the memory is divided into multiple nodes, the default behavior of the system is to allocate memory in the same node as a thread is scheduled to run on. This process works well for small amounts of memory, but when the page pool is more than half of the system memory, it is not physically possible to assign it on single NUMA node. The correct solution to this issue is to interleave the allocated memory across NUMA nodes by using the **`numactl`** command. For IBM Spectrum Scale, this issue can be controlled by `numaMemoryInterleave` parameter, which internally calls **`numactl --interleave=all`**.

Cables

Although much attention is paid to HBA and the switch, the importance of cables is sometimes overlooked. The use of an incorrect cable can result in poor performance or intermittent connectivity problems, which can be difficult to debug.

Therefore, it is best to use supported cables. Consider the switch side and HBA side when considering which supported cable to use.

Cables are available in two types: copper and fiber optic. Each type has its advantages and disadvantages. When you are deciding which type of cable to use, consider the length of the run and the link speed.

Copper cables can be used for a connection within the rack. Consider fiber optic cables for longer distances. Multi-mode fiber cables can be used for connection within a data center. If the connection runs across a building, consider single mode fiber cables because they maintain signal fidelity over considerably longer distances.

2.5.2 OSI Layer 2: Link layer

The IBM Spectrum Scale solutions can be deployed with various layer 2 technologies. This section describes the main technology option that customers can use. The technology choice depends on the following considerations:

- Available bandwidth
- Latency
- Protocol overheads
- CPU use
- Available technical expertise
- Cost

Ethernet

Ethernet is the most popular standard that is used in a data center environment because of its relative simplicity and ubiquitous nature. Ethernet is designed for general purpose, high-speed LAN and is not necessary for a high throughput, low-latency HPC environment. Ethernet technology evolved from 1 Gbps to the present 100 Gbps. The most common Ethernet speeds available are 1 Gbps, 10 Gbps, 40 Gbps, and 25 Gbps, 50 Gbps, and 100 Gbps.

InfiniBand

InfiniBand technology is designed from the ground up for high-throughput and low-latency environments. It started as the network of choice for HPC environments and is gaining traction in enterprise data center environments as well.

InfiniBand offers three different speeds: QDR at 40 Gbps, FDR at 56 Gbps, and EDR at 100 Gbps. More than the speed consideration, the important distinction between the InfiniBand and Ethernet is that InfiniBand offers low latency and low CPU use.

Omni-Path

Omni-Path is a new technology that is offered by Intel. It is designed for high-performance, low-latency, and high-scalability requirements. Omni-Path offers a speed of 100 Gbps.

InfiniBand and Onmi-Path differ in their approach to CPU usage.

All the technologies are evolving in terms of speed and soon will offer 200 Gbps. IBM Spectrum Scale can work on top of any of these technologies (Ethernet, InfiniBand, and Omni-Path) and can use it to the best possible extent.

2.5.3 OSI Layer 3+4

Although the lower layers in the OSI model manage transmitting data over the wires, layers 3 and 4 are responsible for assisting the application to bring the data from the local to the target machine's memory correctly. As part of that process, the data might travel through various "junctions" while traversing different layer 1+2 implementations.

In this section, we describe the common L3+4 technologies that are available today, and how IBM Spectrum Scale solution uses them.

TCP/IP

Although the term *TCP/IP* includes L3 (IP) and L4 (TCP) technologies, it is common to use this term to describe the common Internet Protocol suite.

While in Layer 4, applications often use UDP or TCP. IBM Spectrum Scale, at least for its internal services, is using TCP, mostly because of its reliability features.

The IP feature helps IBM Spectrum Scale traffic to traverse multiple layer 2 virtual LANS (VLANS) by using the routing functions of the IP.

IBM Spectrum Scale traffic can travel between different network logical and physical domains, which allows generic network design methods to be used. The TCP layer features help IBM Spectrum Scale to achieve the distributed application inter-node communication by using "sockets", making sure that the transmitted data arrives to its destination in the correct order, by using the network as best as possible, while taking care of the various issues that might take place in transit (errors, retransmissions, out of order, and so on).

Note: Because of its general availability, IBM Spectrum Scale always uses TCP/IP to create the initial connectivity between nodes. Although further communication might occur by using other networks or other protocols, TCP/IP over the daemon interface is always required.

RDMA

Although TCP/IP provides many efficient services, it also adds CPU overhead because it requires managing multiple OS and network stacks. In some cases, a more efficient mechanism is required to achieve higher throughput and lower latency between the communication end points.

Remote Direct Memory Access (RDMA) provides those requirements by allowing one endpoint to communicate with the other endpoint memory directly, without involvement of the other endpoint's operating system. Therefore, fewer CPU cycles are required.

RDMA depends on layer 2 hardware and drivers to provide the required APIs to operate efficiently. As of this writing, IBM Spectrum Scale supports RDMA on Linux platforms only.

InfiniBand, OPA, and RoCE

RDMA requires layer 2 APIs to be available. Therefore, RDMA support in the IBM Spectrum Scale solution depends on the underlying infrastructure.

The following major technologies provide the required RDMA interfaces for IBM Spectrum Scale to use:

- InfiniBand: Most of the experience with RDMA is on the InfiniBand technology because it was the first common L2 technology to provide those services.
- Omni-Path: Newer fabric from Intel, which provides InfiniBand compatible RDMA interfaces.
- RoCE: RDMA over Converged Ethernet (RoCE) is a recent addition to the RDMA providers because it can use RDMA on top of a relatively regular popular Ethernet infrastructure. Although a few cases exist in which customers deployed this technology, it is still yet to be proven reliable and compatible in multivendor environment, especially regarding its advanced features (routing, multi-port, and so on).

Infrastructure requirements

One of the important considerations when planning an RDMA-based solution is to ensure that the various components (in all layers) are supported by the platforms, operating systems, and cables. Although some of technologies claim to require only end points support (RoCE enabled NICs, for example), support of specific protocols in many cases might be required from all of the network equipment in the path. Other requirements might include end-to-end equipment from a single vendor and other supporting services (subnet managers, connection managers, and so on).

Consult with the relevant provider when designing such fabrics.

TCP/IP and RDMA comparison

When comparing TCP/IP and RDMA technologies, the underlying infrastructure cannot be ignored. Although RDMA promises far more efficient use of the end-point resources (through bypassing the TCP/IP stack), it requires supported infrastructure, including some features that are not provided yet on every switch firmware.

Others might also claim that as CPU power becomes less expensive and CPU offloading of much of the TCP/IP overhead becomes the de facto standard in many NICs. However, it does not make sense to use specialized network equipment to gain the advantages of RDMA.

RDMA in IBM Spectrum Scale today provides the following advantages:

- Fabric aware data routing: Scale supports multiple fabrics, including balancing the traffic across multiple fabrics.
- Traffic “multi-railing”: Assuming multiple ports on a client and a server, IBM Spectrum Scale opens multiple RDMA sessions between them, which better uses multiple ports.

In addition, InfiniBand as a more modern standard still has some layer 2-related advantages, but that discussion is outside the scope of this book.

Mixing multiple technologies (RDMA with TCP/IP)

In many cases, it makes sense to mix the popular nature of TCP/IP with the efficiency and high performing nature of RDMA. Compute nodes and high performing servers are connected to RDMA enabled equipment, while simple workstations might be connected by using plain TCP/IP Ethernet networks.

Although IBM Spectrum Scale supports RDMA and TCP/IP-based environments in parallel, the following issues must be considered:

- Because IBM Spectrum Scale is a parallel file system, client nodes might affect each other. Therefore, a node that is connected to a slow fabric might affect a node that is connected to high-speed fabric.
- The initial negotiation between IBM Spectrum Scale nodes always occurs over the daemon interface by using TCP/IP. Therefore, all nodes that must communicate with each other require the ability to talk to each other over the daemon interface by using TCP/IP.
- Although bridges can bridge between different technologies (InfiniBand to Ethernet, for example), various limitations exists in many cases (the number of concurrent connections, and so on). For more information, contact the specific network provider.
- In most cases, RDMA cannot be used over WAN links. For more information, contact the network provider.
- In many cases, it is advised to implement the following principles when mixing different technologies:
  - Group machines with similar infrastructure to their own cluster. The use of this method might limit the inter-node affects because of the potential slower connectivity on other clusters
  - Consider the use of different protocols for nodes with lower performance requirements or lower connectivity options. For example, use export protocols (such as NFS, CIFS, or object), which are provided by the Cluster Expert Services (CES).

2.6 Network multipathing

Multipathing uses more than one path when talking between two end points. Multipathing features the following major advantages:

- Redundancy: If one path fails, communication can continue with minimal disruption by using an alternative path.
- Performance: The use of multiple paths can create data throughput by sending data over multiple links.

In communication network context, multipathing can be implemented in various layers and locations by using different technologies. Although it is not directly related to IBM Spectrum Scale, redundant and more performing networks are common in IBM Spectrum Scale environments. This section describes several popular network multipathing technologies and what specific consideration IBM Spectrum Scale network usage introduces when the various solutions are implemented.

2.6.1 Layer 2 multipathing

Multipathing in layer 2 is a common and mature practice. Throughout the years, many standards were designed and implemented to achieve multipathing for resiliency and more bandwidth.

Many myths surround those multipathing technologies. In this section, we describe the more popular myths and various related caveats when they are used in IBM Spectrum Scale environments.

Although different operating systems or vendors use different terms for multipathing, one common term that is used in this layer is *link aggregation*.

The common link aggregation technologies that exist in layer 2 do not balance a single connection between multiple ports, mostly to avoid out of order packets and other issues that might degrade performance.

Therefore, performance enhancement that uses link aggregation often is achieved only on many-to-many or many-to-few connection types (that is, many clients talking to many servers or many clients talking to few servers). It has little or no effect when a few-to-few connection type is used because the chances that the small number of connections balance evenly across multiple links is low.

2.6.2 Hosts-to-switch side

Usually, the first place in which a system administrator encounters multipathing is on the connection between a host and the switches. Although different operating systems and device drivers have different names and configuration methods, we focus on the popular Linux-based operating system in this section. Consider the following points:

- Bonding and teaming (LACP)

  In the early days of switch-based networking, few common standards of aggregating multiple links into a single logical pipe were available. Terms, such as *port channel*, *Etherchannel*, and *teaming group* were popular; however, to allow aggregation between different vendors and operating system versions, the Link Aggregation Control Protocol (LACP) was introduced. Its main goal allows two network devices to negotiate an automatic bundling of multiple ports by sending specific packets to the other peer.

- Host side

  Aggregation configuration must be implemented at both connected peers. Linux features the following major aggregation drivers:

  – Bonding

    The bonding driver is the traditional kernel-based aggregation driver. It provides many aggregation modes, the most popular modes being active-passive (which provides only resiliency, but does not require any switch side configuration), and LACP-based mode, which provides resiliency and higher performance, but requires that the switch is configured to support LACP as well.

  – Teaming

    A newer, user space-based link aggregation driver was introduced in RHEL7. It provides all of the original bonding driver modes, allows better manageability, and more flexible control over the aggregation.

Apart from bonding/teaming mode, the administrator should choose which hashing algorithm should be used to decide through which link to transmit for a specific target. For more information about the recommended configurations for IBM Spectrum Scale, see the following resources:

- Chapter 3., “Implementation recommendations” on page 37
- This Red Hat blog entry

2.6.3 Switch side

When LACP aggregation is used, the network switches also must be configured. Although switches from different vendors might use different commands to configure LAG, they often include the need to define which ports are going to be part of the LAG, what hashing to use, and other relevant tunables.

As always when LAG is used, the sender decides the port that is used to transfer a specific stream. The decision to choose the correct hashing algorithm also might be of high importance. For more information, see Chapter 3., “Implementation recommendations” on page 37.

Inter-switch

When LAG is used for hosts (especially with special roles or requirements), the use of LAG between switches is almost mandatory. It is only natural that making sure that the network fabric provides adequate bandwidth and resiliency is a top factor in designing high-speed networks.

LAG/MLAG

Traditionally, it is common to create LAG between two switches. Assuming that both switches adhere to the LACP standard, it should also work between switches from different vendors. Although it can satisfy the high bandwidth and link failure requirements, it cannot protect against switch failure.

Because LAG was limited to 1-1 chassis-host relationship, switch chassis failure causes the LAG to fail. For this reason, MC-LAG (also commonly known as MLAG 6) was introduced. MLAG creates a LAG that spans more than two peers. Therefore, MLAG can have a host that is configured with regular LACP bonding or teaming to communicate with two switches. If one switch fails, the LAG remains operational by using the remaining switch. The same design can be applied to a LAG between two switches.

Note: To implement MLAG, the two participating switches must communicate with each other by using a control channel so that they can form this logically single switch entity.

Officially, the MC-LAG standard limits the participating chassis to two. This amount limits the scalability of MC-LAG for larger networks.

InfiniBand routing

Because InfiniBand is a much newer layer 2 protocol, layer 2 routing was designed and implemented in the InfiniBand standard. Because each endpoint in InfiniBand fabric is known to the entire fabric, smarter routing decisions can be made. For more information about InfiniBand routing, check Openfabric or vendor-specific OpenSM routing documentation.

2.6.4 Layer 3 multipathing

Layer 3 multipathing is implemented by using routing. Routing is a mature and reliable technology that manages multiple data paths inside enterprise networks (LAN and WAN) and the internet.

Traditionally, the main reason for the use of routing was to connect multiple local networks while restricting the type of traffic that moves between those networks. The reasons for the use of multiple networks were often the network size and physical or geographical requirements.

In the past, special devices we need (such as routers) to implement routing. Later on, most L2 switches started to support L3 routing. Therefore, standard switches often also can perform routing.

In parallel, modern workloads required larger environments (cloud infrastructure, large HPC systems, and so on), which require routing to be implemented inside a data center (mostly because of layer 2 multipathing limitations). Therefore, a different way to manage multipathing by using L3 routing started to gain popularity in large environments.

Equal cost multipath routing

Although the use of multipath routing was a common practice for ensuring fault tolerance, the use of it for increased bandwidth is problematic. Also, out-of-order packets (especially in TCP environments) were a common problem. Many other path-related algorithms did not work well when different routes were used. Therefore, the equal cost multipath routing (ECMP) standard emerged.

ECMP provides a deterministic path (when possible) between endpoints that is based on hashing algorithms (similar to the hashing algorithms that are used in LAG). This deterministic path concept solves many of the issues with multipath routing that made it unrealistic. ECMP, with routing protocols, gains popularity in large environments to provide efficient use of the network resources.

OSPF/BGP (routing protocols)

Originally, routing decisions were made based on a “routing tables”. A network or system administrator created a static table that helped a peer (a node or router) determine which path to take to reach some destination network or host. The problem with those static routing tables was that they started to be large and impractical to manage. Therefore, routing protocols were introduced.

Routing protocols are used to build and distribute routing table updates between network routers and multihomed nodes, which reduces the overhead of managing those tables manually.

IBM Spectrum Scale multi-rail

Multi-rail in this context means that the application (IBM Spectrum Scale in this case) takes care of using multiple interfaces or paths in parallel, which achieves higher throughput. Because the application controls the use of multipathing, it can avoid many of the reasons why multipathing cannot be used between two end points, which results in better performance than a single port performance between two end-points.

IBM Spectrum Scale implements multi-railing only when RDMA communication is used (RoCE or InfiniBand). Essentially, two nodes might create multiple RDMA connections by using all available links or paths between the nodes, which results in higher bandwidth.

2.7 Network infrastructure

As a distributed software, IBM Spectrum Scale highly relies on a reliable and high-performing network infrastructure. In most cases, performance and reliability problems that customers encounter are related to the network infrastructure and not necessarily the nodes that are running the software.

Our experience shows that compared to environments that are designed from the ground up for high throughput, it is common that classic IT environments do not have adequate network resources to sustain high throughput data traffic as IBM Spectrum Scale can provide.

In this section, we provide a high-level overview of common practices in designing network infrastructure. It is highly recommended that a customer contact their network consultant to design a network that can provide the required performance.

2.7.1 Common network topologies

Network topology refers to the arrangement of the various network elements, including links, nodes, and switches.

The various optional topologies might introduce, by design, network bottlenecks (also called *blocking factor*). Some are being designed under the assumption that cases in which multiple endpoints attempt to communicate with each other concurrently are rare. In the parallel software-defined storage world (such as IBM Spectrum Scale), concurrent access is a common behavior. The effect of such a configuration might be much higher (in overall performance or reliability of the solution).

Internal switch topologies

Although a network switch might be considered a single element, experience shows that internal switch topologies might affect a network's performance and availability.

Usually, network switches can be divided into the following major types:

- Top of rack switch (TOR)

  Relatively small switches (~40 ports), acting as a single entity. In some cases, the internal components topology introduces some blocking factor (because of mixed speeds, and so on).

- Stackable switches

  To some degree, stackable switches are similar to TOR switches that can be set up to work with other switches as though they are single entity. Stacking introduces easier management (because fewer components are used), but it often introduces a network bottleneck in the stacking channels. It is important to understand the implications of a specific stackable switch in term of throughput, latency, and other potential limits (ARP cache size, and so on).

- Chassis-based switches

  These types of switches are usually based on a base chassis that can host multiple types of “line cards” that provide the connectivity. Those lines cards are connected to each other over the internal chassis backbone.

  The switches often are targeted at large-scale, enterprise-class backbones and include many high availability features. The internal switch backbone is usually wider than stacking links on stackable switches. They also might still introduce bottlenecks that must be considered.

Fat tree

Fat tree network topology is based on hierarchical network structure. The “fat” in the name means that the closer the branch is to the root of the hierarchy, the “fatter” are its links.

The most common architecture is often known as “three-tier topology”, in which the compute nodes are connected to access switches that are connected to an aggregation and distribution layer that is connected to the core layer.[1]

Although the fat tree topology was popular, it has many disadvantages when larger set ups are required:

- Utilization

  The fat tree topology can easily include multiple redundant paths because of the spanning tree protocol. But, those multiple paths cannot be used in parallel; that is, one path cannot efficiently use all the available links.

- Scalability

  As network bandwidth becomes higher, it becomes impractical to introduce a required fat link near the root of the tree. For example, in today’s world where 25 GigE becomes popular and the fastest common path is 100 GigE, every four nodes require 100 GigE or greater to design non-blocking configurations.

  In that case, higher redundancy is required. It becomes two 100 GigE ports per four nodes. For a rack of 84 servers, 168 ports are needed to connect the servers to the TOR, and then another 84 100 GigE ports that act as uplinks.

  This configuration does not make sense.

- Efficiency

  Because communication between clients that are connected to different lower layers (access) must go through the core layer, the latency in this path is higher.

[1] http://www.omnisecu.com/cisco-certified-network-associate-ccna/three-tier-hierarchical-network-model.php

Spine and leaf

Because of the issues with fat tree networks, a new architecture started to emerge, called *spine and leaf network architecture*. As always, it is not such a new topology. This topology uses the Clos network architecture basic principles in creating a highly scalable and efficient network topology.

Compared to fat tree-based approach (which might be considered a "scale up" approach), spine and leaf is more like a scale-out approach because it uses TOR class switches to create a scalable network.

InfiniBand networks were almost always designed to use the spine and leaf topology (and its basic L2 routing supports that design); however, for Ethernet environments, this trend is relatively a new.

For more information, see this web page.

Routing (L2/L3)

It is important to remember that the routing method that is used depends on the network technology and topology that are chosen.

Because it is a newer technology, routing can be managed on layer 2 in InfiniBand networks. On Ethernet (assuming Spanning Tree is not considered routing), routing can be controlled on L3 only.

Usually, InfiniBand L2 routing is easier to implement because it does not require careful subnet and routers design. It is still important to understand problems that might be introduced if nodes are connected to incorrect switches (spines versus leafs), and how to ensure that traffic is balanced between multiple ports. For more information, contact your InfiniBand infrastructure provider.[2]

When Ethernet is used, especially on larger spine and leaf environments, careful routing design is much more complex and a relatively new concept. It is highly recommended to consult with network architects that are familiar with the concept.[3, 4]

2.7.2 LAN and WAN considerations

Apart from the basic network topology design, other factors must be considered when designing a network for distributed software. In many organizations, the following special network equipment can be implemented that might severely affect IBM Spectrum Scale operations:

- Firewalls

  Firewalls are a network security system that creates a barrier between different parts of a network. Firewalls can work on different networking layers (IP layer up to Layer 7 proxies) to protect secured networks from unsecured networks. In many cases, network administrators might implement a transparent firewall so that end devices are unaware that they are communicating through such entity.

[2] https://community.mellanox.com/docs/DOC-2402
[3] https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network
[4] https://tools.ietf.org/html/rfc7938

The following major issues might be encountered when IBM Spectrum Scale traffic (local or remote; that is, WAN replication) is going through a firewall:

  – Blocked ports: You must ensure that a firewall allows traffic on the required ports. For more information about a list of used ports, see the IBM Spectrum Scale documentation.
  – Performance: In many cases, firewalls are focused on security, where performance is secondary. Therefore, a firewall might impose a dramatic effect on the network performance. It is important to verify whether any IBM Spectrum Scale traffic goes through a firewall and what effect you can expect from that specific firewall.

- Virtual Private Network (VPN)

  The VPN is used to transfer private data securely over public networks, such as the internet. It often uses many authentication and encryption technologies to achieve that level of security.

  Although VPN presents the network as though it is a standard private network, it encapsulates the communication packets largely affecting its performance.

  No specific reasons exist that prevent IBM Spectrum Scale from working over VPN lines. Still, it is important to consider the performance and reliability effect of this technology when designing a network for IBM Spectrum Scale.

- WAN optimizers (WAN accelerators)

  WAN optimization products are often used across WAN links to better use those relatively slow and unreliable links for data transfer. In theory, they should help IBM Spectrum Scale work better on those environments. However, experience shows that in some cases, they cannot handle the load and have a negative effect on system performance.

2.7.3 Network monitoring tools

Managing a complex and demanding network is a challenging task. In most network-related issues that affect IBM Spectrum Scale, the problems are discovered to be issues with the network fabric (and not necessarily with the endpoints). The issue might be a mis-configured switch, wrong routing rules, broken cable causing communication errors, and so on.

It is advised to use some network monitoring tools to ensure that the network is healthy, and to make the task of troubleshooting network issues easier.

IBM Spectrum Scale provides extensive endpoint monitoring (for example, interface errors), but it cannot monitor the network fabric, especially with issues that occur only when multiple nodes are talking to multiple nodes through multiple paths.

Many network monitoring tools are available commercially and free of charge. Some tools are related to specific technologies (such as UFM for InfiniBand) and other tools are more generic and can monitor different technologies.

2.7.4 Flow control techniques

In data communications, flow control is the process of managing the rate of data transmission between two nodes to prevent a fast sender from overwhelming a slow receiver.

Numerous flow control technologies are available. In this section, we focus on the technologies that are relevant for common IBM Spectrum Scale environments, most notably TCP/IP, and RDMA and their relevant layer 2 transports.

Flow control can be managed in multiple layers. The higher the layer, the better its understanding of what is occurring, which allows for finer control.

Layer 2 flow control

The most common flow control technologies that are used are at layer 2. InfiniBand and Ethernet have flow control implementation, InfiniBand flow control[5] is mostly not transparent to the user because of InfiniBand efficiency (we focus on Ethernet in this section).

Ethernet flow control can often be divided into two major types: Pause frame (that is, Global Pause) and Priority Flow Control (PFC).

By definition, Ethernet is an unreliable network standard. Therefore, it is not guaranteed that a packet reaches its destination. If the receiver cannot receive, it drops the packets and the higher layer protocols handle it. But, in some cases, the cause for the overload is the Ethernet layer. The following flow control technologies were added in this layer:

- Pause Frame (802.3x)

  Ethernet pause frame (that is, global pause) means that a receiver sends a special multicast message telling the sender to pause for a specific period before sending again. As a result, it throttles its received content.

  The major disadvantages of global pause are that *everything* is stopped, which prevents all available bandwidth from being used.[6]

- Priority Flow Control (PFC, 802.3Qbb)

  Because of the limitations of global pause, and other protocols that require lossless (or near lossless) transport mechanisms were introduced (most notably FCoE, RoCE, and so on), a better standard was required. PFC extended the single class in Global Pause to eight different classes, each of which can be controlled independently from others. As a result, a specific traffic or connection can be paused without directly affecting the others. It allows for the coexistence of protocols that require lossless transport with protocols that can work on lossy networks.

Layer 3 + 4 flow control

TCP/IP features a built-in mechanism that is used to manage network congestion. It is using dropped packets as a symbol that something is wrong, and then tries to use various algorithms to find the correct rate.

This mechanism is a bit of a brutal method of “let’s see when it breaks – and then ask”. Therefore, Explicit Congestion Notification (ECN) extension was introduced. ECN allows the receiver a much more graceful way of signaling the sender that it must slow down without dropping packets, which allows better management by the endpoints.[7]

When RDMA is used, it expects a mostly lossless network. On InfiniBand, as a lossless layer 2 network technology, it is easy. On Ethernet, an efficient flow control mechanism is required when RDMA is used over converged Ethernet (RoCE)[8].

[5] https://www.mellanox.com/pdf/whitepapers/IB_Intro_WP_190.pdf
[6] https://theithollow.com/2013/05/07/flow-control-explained
[7] https://community.mellanox.com/docs/DOC-2022
[8] https://community.mellanox.com/docs/DOC-1451

Quality of service

Quality of service (QoS) provides different service levels to different applications. It complements flow control to better manage traffic. Although on InfiniBand and RoCE some QoS exist, such as definitions in Scale, no direct interaction occurs between Scale and network QoS. Still, IBM Spectrum Scale traffic can be prioritized by using standard network infrastructure (based on port number, iptables tagging, and so on).

Network services

If network services are part of the network infrastructure or belong to the application level, we describe the following services that IBM Spectrum Scale might rely on:

- Domain Name System

  Every well-designed network includes a highly available and scalable name resolution service, such as DNS. Scale (as most other software) uses name resolution and reverse name resolution.

- Network Time Protocol (NTP)

  To ensure that all network elements time is in sync, NTP often is used. It is important for various services to have synced time on all nodes, such as authentication and logging.

- Key management[9]

  When key-based services are used (often for authentication, encryption, and so on), a highly available key management service is required. For example, when IBM Spectrum Scale native encryption feature is used, failing to read a key from a key server results in an I/O error.

- Authentication servers

  To connect to a system or a service, it is common to use some central authentication servers. Different protocols might support different authentication protocols. For more information, see the IBM Spectrum Scale documentation.

[9] http://www.redbooks.ibm.com/abstracts/redp5384.html

2.8 Network considerations for special use cases

In this section, we describe some network considerations for special use cases.

Mixing network IBM Spectrum Scale daemon network

IBM Spectrum Scale supports the use of multiple networks for daemon communication within a cluster and when connecting remote clusters. The subnets functions enable customers to have a high-speed network connection between a few nodes, but all of the nodes within a cluster must have at least one Internet Protocol network address space in common (see Figure 2-7).

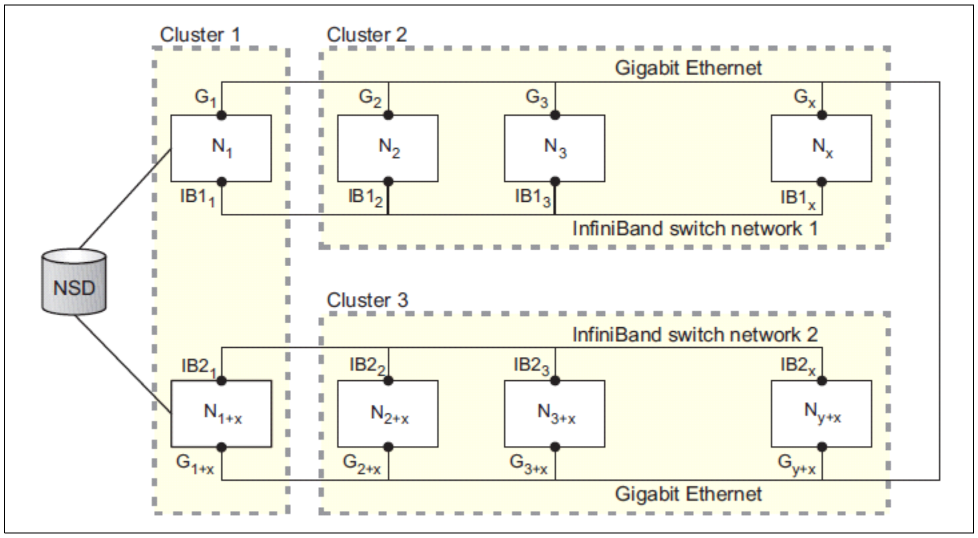


Figure 2-7 Cluster communication

When configuring an IBM Spectrum Scale cluster, you start by defining an IP address that can be used for all nodes to communicate. You specify this IP address when a cluster is created or a node is added to a cluster. This IP address is displayed when you run the **`mmlscluster`** command. This IP address must be in a TCP/IP address space that allows all of the IBM Spectrum Scale nodes within the cluster or across multiple clusters sharing data to communicate by using TCP/IP.

Then, you can define other Internet Protocol networks that are used for node-to-node communications. These other networks are defined by using the subnets parameter.

The subnets operand on the **`mmchconfig`** command allows you to specify an ordered list of networks that are available to IBM Spectrum Scale for daemon communications. Multiple subnets can be used within a cluster and when communicating between IBM Spectrum Scale clusters. When communicating with other IBM Spectrum Scale clusters, each subnet that is listed can include a list of cluster names that specifies other IBM Spectrum Scale clusters that can access the same subnet.

When the IBM Spectrum Scale daemon starts on a node, it builds a list of its own networks from its local configuration and then listens on all of the networks it has that are in the subnets list and the cluster shared IP address.

When another IBM Spectrum Scale node wants to establish a connection with this node, it checks whether it is on one of the subnets that is specified on the subnets configuration parameter. If that subnet is specified, the node picks the first subnet in the list that it can use to communicate with the node.

If the shared IP address for the node (which was specified when the cluster was created or when the node was added to the cluster) is not specified in the subnets configuration parameter, IBM Spectrum Scale automatically adds it to the end of the node's IP address list. When a node is uses the shared IP addresses, the subnet does not need to be listed in the subnets configuration parameter.

For example, a common way to configure a system is to use the hostname that resolves to the shared Ethernet IP address for the node in the **`mmcrcluster`** command. Then, if you want to use a high-performance network for communication between nodes within the cluster, the high-performance network is added to the list of networks in the subnets configuration parameter. You can add multiple entries in the subnets configuration parameter, which ensures that they are used in that order.

When a node joins a cluster (its own cluster on start, or another cluster when mounting a file system owned by another cluster), the node sends its list of IP addresses (ordered according to the order of subnets configuration parameter) to the cluster manager node, which forwards the list to all other nodes as part of the join protocol. No other information must be propagated.

When a node attempts to establish a connection to another node, IBM Spectrum Scale determines the destination network address to use according to the following procedure:

1. For each of its own networks, it searches the other node's list of network addresses for an address that is on the same subnet. For shared IP addresses, this process is done by comparing IP address and subnet mask values for its IP address.

   For private IP addresses, IBM Spectrum Scale assumes that two IP addresses are on the same subnet only if the two nodes are within the same cluster, or if the other node is in one of the clusters that is listed in the subnets configuration parameter. Private IP addresses are defined as being in a specific address range (IP addresses on a 10.0.0.0, 172.16.0.0, or 192.168.0.0 subnet) as defined in RFC 1597.
2. If the two nodes have more than one network pair on a common subnet, IBM Spectrum Scale uses the first pair that is found according to the order of subnets that is specified in the initiating node's configuration parameter.
3. If no two networks are on the same subnet, IBM Spectrum Scale uses the last entry in each node's network address list. That is, the last subnet that is specified in the subnet's configuration parameter is assumed to be on a network that is accessible from all nodes in the cluster.

The subnets functions does not provide high availability. The communication network is decided when the IBM Spectrum Scale daemon starts. If that network fails, the communication does not automatically switch to the next available network in the subnets list.

3

Implementation recommendations

This chapter describes recommended configuration settings, tools, and commands that are available for network configuration in IBM Spectrum Scale or IBM Elastic Storage System environment. Before starting with implementation, various considerations are required for optimal network design and planning the environment. It is recommended that you complete the design and planning as described in Chapter 2, “Network planning and best practices for IBM Spectrum Scale and IBM Elastic Storage System” on page 9.

You can check whether various elements of network infrastructure (that is, network adapters, network cables, network switches, network bandwidth and latency) are available per the solution design.

Following the recommended network design and implementation practices can prevent any need to troubleshoot IBM Spectrum Scale issues that might not look as though they are related to network environment.

This chapter includes the following topics:

- “Network verification tools” on page 38
- “Ethernet Network Adaptor configuration settings” on page 38

3.1 Network verification tools

This section describes various network verification tools that you can use before starting implementation. It is recommended that you use these tools before you start to configure IBM Spectrum Scale or IBM Elastic Storage System configuration because it is much easier to run these tools when no active workload is running over network in the IBM Spectrum Scale environment.

3.1.1 IBM Spectrum Scale mmnetverify command

The **mmnetverify** command is highly recommended to be used to verify the network configuration and operation of a group of nodes before you organize them into an IBM Spectrum Scale cluster or before you add nodes to an IBM Spectrum Scale Cluster. The command requires that all the nodes under network verification have IBM Spectrum Scale software installed and all the nodes can remote shell to other nodes without the use of a password.

While verifying the network with the **mmnetverify** command, ensure that you check the network between all nodes within the cluster, especially the following combinations:

- Between all IBM Spectrum Scale NSD server nodes or IBM ESS I/O server nodes
- Between all nodes with manager roles or manager license and NSD or I/O server nodes
- Between each client node and all manager/I/O server/NSD server nodes
- Between any protocol servers, backup servers and all NSD server/client nodes
- Between NSD or I/O servers at other sites in case of synchronous or AFM replication
- Between NSD or I/O servers and quorum nodes in case of synchronous replication

For more information about usage with examples, see 4.4, “Verifying network with IBM Spectrum Scale mmnetverify command” on page 53.

3.1.2 IBM Spectrum Scale nsdperf tool

The **nsdperf** tool is highly recommended to be used to verify the network performance and characteristics within a group of nodes before you organize them into an IBM Spectrum Scale cluster. You must install this tool on all nodes to verify the network performance between the nodes.

Similar to the **mmnetverify** command, it is recommended to run the **nsdperf** tool between the all nodes within the cluster, especially within the combinations, as described in 3.1.1, “IBM Spectrum Scale mmnetverify command”.

For more information about usage with examples, see 4.5, “Testing network performance with IBM Spectrum Scale nsdperf tool” on page 59.

3.2 Ethernet Network Adaptor configuration settings

This section describes various network configuration guidelines at the physical and link layer on the IBM Spectrum Scale/IBM ESS server and client nodes.

3.2.1 Ring buffers

Ring buffers setting on the Network Interface Cards (NIC) indicate the number of frames that NIC can store in its internal buffer. Ring buffers are important to handle bursts of incoming packets, especially if some delay exists in handling the incoming packets at the Kernel level.

NIC ring buffer sizes vary per NIC vendor and NIC grade (that is, server or desktop). The NIC Ring buffers are usually configured at less than supported maximum values. By increasing the Rx/Tx ring buffer size, you can decrease the probability of discarding packets in the NIC during a scheduling delay. For more information, see 4.9, "Understanding and resolving the effect of network packet loss" on page 74.

Example 3-1 shows how to check and set ring buffers.

Example 3-1 Checking and adjusting ring buffers

```
# Check current ring buffers setting through ethtool
 ethtool -g <interface>
Ring parameters for enP2p1s0f0:
Pre-set maximums:
RX:   8192
RX Mini:0
RX Jumbo:0
TX:   8192
Current hardware settings:
RX:   1024
RX Mini:0
RX Jumbo:0
TX:   1024

# Adjust the size of ring buffers using ethtool
ethtool -G <interface> rx 8192 tx 8192
```

Note: Consider the following points:

- To make the change persistent, you can add it to the `/sbin/ifup-local` script, which is called by `/etc/sysconfig/network-scripts/ifup-post` as the interfaces becomes active. You also can add this script to another script that is called by tuned (the daemon process that is available on Linux) after the system comes up.
- If created a bond that consists of subordinate interfaces, the ring buffer configuration change must be applied to the subordinate interfaces.

3.2.2 Maximum transmission unit

The main advantage of using jumbo frames is that it reduces the CPU overhead that is required for TCP processing and thereby provides optimum network utilization and higher throughput. Because jumbo frames are larger than standard frames, fewer frames are needed and therefore, CPU processing overhead is reduced.

When sending and receiving large data, enabling jumbo frames improves overall network performance. A maximum transmission unit (MTU) size of 9000 is recommended for IBM ESS/IBM Spectrum Scale.

Note: You must ensure that all the components in your network infrastructure (sender and receiver NIC, switches, and so on) are enabled for jumbo frames.

The following example shows the command to set MTU to value 9000:

```
# Set MTU size to 9000 using ifconfig
    ifconfig enP2p1s0f0 mtu 9000
```

Note: If you created a bond that consists of subordinate interfaces, the MTU configuration must be applied to each of the subordinate interfaces. You can apply these changes by using `ifconfig` or an NIC card-specific utility on the system.

3.2.3 Transmission queue length

On high-speed networks with low latencies, it recommended you set transmission queue length (txqueuelen) to a value higher than default of 1000:

```
# Set txquelen using ifconfig
 ifconfig enP2p1s0f0 txqueuelen 10000
```

3.2.4 Flow control

Flow control is a feature that is defined in the IEEE 802.3x specification. It enables a receiving device to signal congestion to a sending device, which allows for the sending device to temporarily halt transmission. This process alleviates congestion at the receiving device.

For IBM Spectrum Scale and IBM ESS networks, it is recommended that you enable flow control end to end from the NSD server to the NSD client, including all network devices, such as NICs, switch ports, and switches.

You can check whether flow control is enabled for the receive and transmit on your NIC cards by using the **`ethtool`** command. Example 3-2 shows how to set the **`ethtool`** command to check and enable flow control on Linux host.

Example 3-2 Checking for receive and transmit flow control setting

```
# Check receive and transmit flow control setting
ethtool -a enP2p1s0f0
Pause parameters for enP2p1s0f0:
Autonegotiate: off
RX: off
TX: off
```

As shown in Example 3-2, flow control is disabled for `enP2p1s0f0` on the node. To enable flow control, run the **`ethtool -A enP2p1s0f0 rx on`** command.

Network switches support different types of flow control mechanisms and are better suited to handle the flow control. You can enable flow control, such as pause frames, on your network switches for transmit and then configure all nodes in the cluster to receive and act on receiving pause frames.

3.3 TCP/IP settings

This section describes various TCP/IP-related tuning parameters that are recommended to be set on all nodes in the cluster that are using TCP/IP for IBM Spectrum Scale communication between the nodes.

3.3.1 TCP/IP send and receive buffers

TCP/IP buffer-related settings should be configured on all nodes in the IBM Spectrum Scale/IBM ESS cluster. You can use higher values (16M), as shown in Example 3-3, on the NSD/I/O servers and lower values (4M and 8M) on the clients.

Example 3-3 TCP/IP buffer settings

```
# max socket buffer size
net.core.optmem_max=16777216

# max socket read buffer size
net.core.rmem_max=16777216

# max socket write buffer size
net.core.wmem_max=16777216

# default socket buffer read size
net.core.rmem_default=16777216

# default socket buffer write size
net.core.wmem_default=16777216

# TCP receive buffer mem (min, default, max)
net.ipv4.tcp_rmem=4096 87380 16777216

# TCP send buffer mem (min, default, max)
net.ipv4.tcp_wmem=4096 65536 16777216
```

You can set these values in `/etc/sysctl.conf` file or by using a tuned profile.

3.3.2 TCP/IP window scaling

The following TCP/IP latency and window scaling-related settings should be configured on all nodes in the IBM Spectrum Scale/IBM ESS cluster:

```
# Give preference to low latency over higher throughput
net.ipv4.tcp_low_latency=1

# Determine space to be used for window vs application buffer
net.ipv4.tcp_adv_win_scale=2

# RFC 1323 - support TCP window sizes larger than 64K
net.ipv4.tcp_window_scaling=1
```

3.3.3 TCP/IP settings for network with packet loss

The following TCP/IP settings are recommended when you expect some packet loss between NSD server and client nodes because of network congestion or other reasons. When these settings are used in a protocol environment, ensure that all protocol clients are running suitable TCP stacks where these settings are implemented and do not have any issues.

These settings behave differently depending on TCP stack versions on different nodes in the IBM Spectrum scale cluster and network infrastructure delays. You can start with these settings as disabled and enable them in case you notice many retransmissions1;

```
# To Enable selective ack
net.ipv4.tcp_sack=1

# To Enable timestamps
net.ipv4.tcp_timestamps=1
```

3.3.4 Other TCP/IP setting

In addition, the following TCP setting should be configured on all nodes in IBM Spectrum Scale/IBM ESS cluster:

```
# Mex packets to be queued at interface layer
net.core.netdev_max_backlog=250000
```

3.4 Remote Direct Memory Access settings

Typically Remote Direct Memory Access (RDMA) communication is used for a small cluster or communication between a set of nodes within a cluster. Typically, RDMA is configured between NSD Server and I/O server nodes or between NSD server and a small group of client nodes that has high-speed network infrastructure (InfiniBand or Ethernet). This section describes various RDMA-related tuning parameters that are recommended to be set on all nodes in the cluster that are using RDMA for IBM Spectrum Scale communication between the nodes.

3.4.1 Settings to enable RDMA between nodes

It is recommended that you enable RDMA settings only for those nodes that use RDMA for communication. A node that has RDMA enabled determines whether the peer node also has RDMA enabled and if not, starts using TCP/IP communication:

```
# Enable InfiniBand Rdma for all Power nodes
# mmchconfig verbsRdma=yes -N gss_ppc64

# Enable use of InfiniBand Rdma for all traffic between nodes
# mmchconfig verbsRdmaSend=yes -N gss_ppc64
```

For large clusters (> 100 nodes), enable the **`verbsRdmaSend`** option only if all nodes in the cluster are at IBM Spectrum Scale version 5.0 or above.

3.4.2 Tuning for RDMA

IBM Spectrum Scale provides the following methods to scale the data transfer between the nodes over RDMA.

- The **verbsRdmaPerConnection** configuration parameter sets the maximum number of data transfers that are allowed at the same time per connection. The default value for this setting is 8; whereas this value can be changed to higher number (for example, 14 for larger cluster and also depending upon how much data is being transferred over RDMA between the nodes).
- The **verbsRdmasPerNode** configuration parameter sets the maximum number of simultaneous data transfers that are allowed at the same time per node. By default, the value of **nsdMaxWorkerThreads** is used. You can increase the value of **nsdMaxWorkerThreads** or this setting to scale up the transfer between the nodes.

3.5 Network multipathing settings

This section describes various host side (Linux) multipathing-related configurations that are recommended for the IBM Spectrum Scale/IBM ESS environment. Other than host-side configuration, you must also configure your switch for a multipathing configuration that is consistent with your host side configuration.

The recommended configuration for achieving redundancy and performance together is to have two dual ported NIC cards in a bond + team configuration on host side and with MLAG configured on the switch side. You also must suitably connect one port from each NIC card to one switch and connect other ports from each of the NIC card to the other switch that is configured in the MLAG configuration. The similar configuration is recommended on all clients and NSD server nodes. The IP address that is configured on the bonded interface should be the IP address that is used to configure the IBM Spectrum Scale daemon communication.

Selecting LACP hash algorithm

IBM Spectrum Scale opens a single connection between nodes in the cluster, including an IBM Spectrum Scale client and an NSD server. The selection of the hash algorithm depends on the following factors in the cluster:

- Number of NSD server nodes

  If you have a limited number of NSD server nodes, you must ensure that connections to different NSD servers are well distributed among the subordinates that are configured on the system. You can get such ensured or fixed distribution through layer 2+3 hash algorithm, which always sends traffic to a specific peer IP over a specific subordinate only.

- Number of client nodes

  When you have a limited number of clients (for example, a single six-node protocol cluster), it is recommended that you configure layer 2+3 hash. If you have many clients, it is recommended that you configure layer 3+4 hash algorithm. With large number of clients, you are likely to get even distribution by using a hash algorithm of layer 3+4 and do not need to be concerned about the exact distribution across subordinate nodes. However, the distribution is likely to be balanced because of the large number of clients.

You must configure the same hashing algorithm on network switches and nodes in IBM Spectrum Scale cluster. The `xmit_hash_policy` bonding parameters that you placed in the **BONDING_OPTS** field of the **ifcfg-bondX** file in Linux operating system should align with the network switch hashing algorithm, which is also called the *port channel load-balancing policy*.

4

Network monitoring and troubleshooting

IBM Spectrum Scale stability and performance is highly dependent on the underlying networking infrastructure, and many IBM Spectrum Scale problems are resolved by addressing such infrastructure issues. The references to *the network* in this paper include this wide definition of the network stack.

This chapter includes the following topics:

4.1 Network-related issues overview

Table 4-1 lists some examples of network issues that are related to IBM Spectrum Scale and maps these examples to the layers that are defined by the OSI model of computer networking.

Table 4-1 Potential network issues with IBM Spectrum Scale

OSI Model	Example
Application	► IBM Spectrum Scale and the applications that use IBM Spectrum Scale can encounter "issues" at this layer. ► Recent IBM Spectrum Scale versions contain "network-related" fixes, which improve the code's performance and resiliency; for example, IBM Spectrum Scale now prioritizes the sending of certain Remote Procedure Calls (RPC) to avoid an expels under heavy load conditions. ► IBM Spectrum Scale has many network-related configuration options and sometimes tuning at this layer might required to achieve optimal results.
Presentation	IBM Spectrum Scale problems can be experienced because of problems at this layer; for example, when external security services (such as TLS and Kerberos) are configured, problems with these services can result in IBM Spectrum Scale functional failures, such as expels or the inability to access data.
Session	IBM Spectrum Scale RPC layer issues can be considered as "Application layer" but note that the OSI model does not always lend itself to clear layer definitions (the exact OSI layering of TLS, ARP, ICMP, and so on, can be discussed).
Transport	► Firewall configuration problems, or bug in the TCP/IP stack, are examples of problems that can affect IBM Spectrum Scale at this layer; for example, IBM Spectrum Scale TCP ports (1191s by default) must not be blocked by a firewall. ► During problem determination, reviewing TCP data structures that are associated with sockets that are used by IBM Spectrum Scale (for example, RTO, and send and receive socket queues) can isolate if a problem is specific to the Transport or a lower layer (Network, Data, or Physical layer)
Network	IBM Spectrum Scale issue, such as an expel that is caused by a misconfiguration in IP routing; IBM Spectrum Scale needs full TCP/IP connectivity between all nodes in a specific cluster so that if any routing protocols are used, they must be correctly configured to avoid performance problems or functional failures.
DataLink	► IBM Spectrum Scale performance problems occur if flow control is misconfigured. ► Ensuring that switch flow control is correctly configured in appropriate cases can avoid IBM Spectrum Scale performance issues and expels. ► ARP resolution issues can lead to expels and performance problems; therefore, sysctl tuning for ARP-related settings is needed on large systems.

OSI Model	Example
Physical	► The choice of hardware defined IBM Spectrum Scale performance expectations. ► Broken or bad network-related hardware (for example, cables, interposers, adapters, and switches) can lead to (sometimes intermittent) expels or performance. ► Hardware configuration design choices; for example, choosing the blocking factor of network the network (such as oversubscribed leaf or spine links) can affect performance.

Network issues can be intermittent and show varying symptoms, span different networking layers, and affect only a subset of adapters, nodes, switches, and so on. Debugging these problems might require a wide range of approaches and tool sets. The set network-related tools and commands that are described in this chapter can be divided into the following categories:

- IBM Spectrum Scale tools and commands
- Linux tools and commands

4.2 Network monitoring with IBM Spectrum Scale GUI

The IBM Spectrum Scale GUI provides a way to do both network monitoring and performance analysis. On a cluster that configured the IBM Spectrum Scale GUI, the GUI provides a good starting point for analyzing and troubleshooting performance issues in real time.

4.2.1 Monitoring Network Events Using the GUI

When checking to see if a cluster is experiencing a network-related issue, administrators can check what events are reported in the IBM Spectrum Scale GUI. The GUI displays a list of all network-related events as reported by the IBM Spectrum Scale System Health subsystem (command-line access is available by running the **`mmhealth`** command).

For more information about monitoring events by using GUI see IBM Knowledge Center.

For more information about networking events that are addressed by the GUI, see the Network events and CES network event tables in the “Events” section of Chapter 36: References, in *IBM Spectrum Scale Version 5.1.0 Problem Determination Guide*.

4.2.2 Monitoring Network Performance Using the GUI

From IBM Spectrum Scale 5.0 onwards, the GUI is enhanced to allow for monitoring network performance statistics, health, and configuration details. In addition to providing health event details, the GUI provides a graphical view of overall IP and RDMA throughput.

To aid in the analysis of performance or functional issues, network interface throughput can be monitored by way of time-series graphs, which are refreshed on an interval that is defined by the administrator. This data can be used, for example, to correlate an unexpected change in the network traffic with another issue, such as a performance problem occurring without any change in the workload running.

The GUI also gives the capability to view all the network interfaces that are part of the Ethernet and InfiniBand networks in the cluster. It also allows you to view performance details in graphical format or to see any events that are reported against individual adapters.

For more information about monitoring networks by using GUI in IBM Spectrum Scale 5.1, see IBM Knowledge Center.

4.2.3 Network Monitoring with IBM Spectrum Scale mmhealth command

The **mmhealth** command monitors the health status of a node and services that are hosted on the node. You can use the **mmhealth** command to view the health status of a whole cluster in a single view. From a network perspective, **mmhealth** can list network-related events and the status of the network.

The **mmhealth** network monitor capability automatically checks and analyzes the current network configuration and reports all issues as events in a centralized system health interface. When a network issue is suspected, a good starting point is to run the following **mmhealth** query:

```
mmhealth node show network -v
```

The **mmhealth** command (see Example 4-1) helps detect and show the following information for the TCP/UP and RDMA network:

- Which network interfaces are relevant for IBM Spectrum Scale
- Whether these NICs are up or down
- The physical state of the links of these NICs
- The bonding structure and to compute and output the resulting state (for example, if all subordinates of a bond are DOWN, the bond is DOWN, but if any of them comes back online, it is only DEGRADED)
- Whether too many TX errors appear by a NIC; that is, the communication channel is damaged (for example, a bad contact)
- Whether a node becomes inaccessible for the IBM Spectrum Scale administration routines

Example 4-1 mmhealth command example

```
# mmhealth node show network -v

Node name:               ch-41.localnet.com

Component                Status                   Reasons
-------------------------------------------------------------------
NETWORK                  HEALTHY                  -
  eth0                     HEALTHY                  -

Event                    Parameter                Severity                 Description
-----------------------------------------------------------------------------------------------
network_connectivity_up  eth0                     INFO                     NIC eth0 can connect to the gateway
network_link_up          eth0                     INFO                     Physical link of the NIC eth0 is up
network_up               eth0                     INFO                     NIC eth0 is up
no_tx_errors             eth0                     INFO                    NIC eth0 had no or an insignificant number of
TX errors

Error Case

[root@ch-41 ~]# mmhealth node show network

Node name:               ch-41.localnet.com
```

```
Component                Status                   Reasons
-------------------------------------------------------------------
NETWORK                  HEALTHY                  network_down
  eth0                     HEALTHY                  network_down

Event                    Parameter                Severity                 Description
-------------------------------------------------------------------------------------------------------
network_down             eth0                     ERROR                    NIC eth0 is down
```

The **mmhealth** command also provides information regarding any IBM Spectrum Scale CES network events, which is useful when protocols are configured.

For more information about networking events that are addressed by **mmhealth** command, see the Network events and CES network event tables in the “Events” section of Chapter 36: References, in *IBM Spectrum Scale Version 5.1.0 Problem Determination Guide*.

For more information about the **mmhealth** command, see IBM Knowledge Center.

4.2.4 IBM Spectrum Scale call back commands to monitor network-related events

The IBM Spectrum Scale callback feature (as described in IBM Knowledge Center) allows for event driven responses to be run in response to specific conditions, which results in efficient notification and data collection mechanisms. Although no callbacks are specific to network events, callbacks can be enabled to detect events, such as “nodeLeave”, or 'unmount', which can be related to network failures.

Also, the “deadlockDetected” event might signal a performance issue, which makes this trigger potentially useful for performance-related data collection (the deadlock event refers to a long thread wait time condition and not a true deadlock condition).

The callback mechanism can be used to notify users of events and to collect debug data for such events. For example, a debug data collection script can be enabled to run after each local unmount of a IBM Spectrum Scale file system, as described next.

First, a script to collect data is created that runs on each unmount of an IBM Spectrum Scale file system. In this example, we assume that we created such a data collection script, copied it to `/tmp/unmount_data_collection` on all nodes, and gave the script run permission.

Then, we enable this script to run on each unmount event by running the **mmaddcallback** command:

```
mmaddcallback unmount_callback --command /tmp/unmount_data_collection --event
unmount
mmaddcallback: Propagating the cluster configuration data to all affected
nodes.  This is an asynchronous process.
```

4.3 Network troubleshooting with IBM Spectrum Scale mmdiag command

The **mmdiag** command can be used to query various IBM Spectrum Scale internal state details for troubleshooting and tuning purposes. The command obtains the required information by querying the IBM Spectrum Scale daemon process (**mmfsd**), and thus functions only when the IBM Spectrum Scale daemon is running. From a network perspective, **mmdiag** features the following three options that must be considered for troubleshooting:

- `mmdiag --network`

  Displays information about **mmfsd** network connections and pending messages (see Example 4-2). This command provides a production safe way of collecting low-level socket state data on the connections between **mmfsd** processes, and the messages on which the IBM Spectrum Scale daemon is waiting.

Example 4-2 mmdiag command example

```
mmdiag --network
=== mmdiag: network ===

Pending messages:
  (none)
Inter-node communication configuration:
  tscTcpPort      1191
  my address      9.114.53.217/25 (eth2) <c0n2>
  my addr list    9.114.53.217/25 (eth2)
  my node number  4
TCP Connections between nodes:
  Device null:
    hostname                      node     destination     status     err  sock  sent(MB)  recvd(MB)  ostype
    c941f1n05.pok.stglabs.ibm.com  <c0n1>   9.114.78.25     broken     233  -1    0         0          Linux/L
  Device eth2:
    hostname                      node     destination     status     err  sock  sent(MB)  recvd(MB)  ostype
    c941f3n03.pok.stglabs.ibm.com  <c0n0>   9.114.78.43     connected  0    61    0         0          Linux/L
    c870f4ap06                    <c0n3>   9.114.53.218    connected  0    64    0         0          Linux/B
Connection details:
  <c0n1> 9.114.78.25/0 (c941f1n05.pok.stglabs.ibm.com)
    connection info:
      retry(success): 0(0)
  <c0n0> 9.114.78.43/0 (c941f3n03.pok.stglabs.ibm.com)
    connection info:
      retry(success): 0(0)
      tcp connection state: established     tcp congestion state: open
    packet statistics:
      lost: 0     unacknowledged: 0
      retrans: 0     unrecovered retrans: 0
    network speed(µs):
      rtt(round trip time): 456     medium deviation of rtt: 127
    pending data statistics(byte):
      read/write calls pending: 0
      GPFS Send-Queue: 0     GPFS Recv-Queue: 0
      Socket Send-Queue: 0     Socket Recv-Queue: 0
  <c0n3> 9.114.53.218/0 (c870f4ap06)
    connection info:
      retry(success): 0(0)
      tcp connection state: established     tcp congestion state: open
    packet statistics:
      lost: 0     unacknowledged: 0
      retrans: 0     unrecovered retrans: 0
    network speed(µs):
      rtt(round trip time): 8813     medium deviation of rtt: 13754
    pending data statistics(byte):
      read/write calls pending: 0
      GPFS Send-Queue: 0     GPFS Recv-Queue: 0
      Socket Send-Queue: 0     Socket Recv-Queue: 0
Device details:
  devicename     speed     mtu       duplex     rx_dropped rx_errors tx_dropped tx_errors
  eth2           1000      1500      full       0          0         0          0
diag verbs: VERBS RDMA class not initialized
```

Breaking data down in Example 4-2 by sections, first the "Pending messages" shows details regarding messages that are outstanding (for example, the thread that sent the message, the destination, and how long the message was pending).

Example 4-3 shows pending messages that are displayed before an RPC timeout node requested expel is about to occur (an expel should result when a pending `commMsgCheckMessages` message reaching the-default-300 second timeout).

Example 4-3 Pending messages example

```
Pending messages:
  Message ID 362, type 13 'nsdMsgReadExt' age 348 sec
    sent to 1 nodes, waiting for reply from 1 nodes
    sent by thread 'BackgroundSyncThread' (0x7F493423E630)
    dest XX.XX.2.27       status pending   , err 0, reply len 0 by TCP connection
  Message ID 363, type  9 'sgmMsgSGClientCmd' age 345 sec
    sent to 1 nodes, waiting for reply from 1 nodes
    sent by thread 'CommandMsgHandlerThread' (0x7F4934F51580)
    dest XX.XX.2.27       status pending   , err 0, reply len 0 by TCP connection
  Message ID 370, type  3 'commMsgCheckMessages' age 299 sec
    sent to 1 nodes, waiting for reply from 1 nodes
    sent by thread 'EEWatchDogThread' (0x7F4934F68D40)
    dest XX.XX.2.27       status pending   , err 0, reply len 0 by TCP connection
```

In Example 4-3 in which messages are pending to `XX.XX.2.27`, the next debug step likely is to review the connection details for this host.

After the pending messages, a list of connections is displayed, which is useful for checking how much data was exchanged between the current daemon connections (`sent(MB)` and `recvd(MB)`), and validating all the connection states.

Next, the connection details section provides low-level TCP and verbs level data, and the state of the socket send and receive queues. For the previous example in which messages are pending to `XX.XX.2.27`, in this section we likely want to check the "tcp congestion state" (generally, it should be in "open" state), and the "retrans" value - if "retrans" is consistently nonzero that indicates retransmission that causes performance of functional (for example, expel) issues.

Optionally, as a debug approach to network or performance issues, other low-level socket data, such as the network round-trip time (in microseconds), is available.

- `mmdiag --rpc <options>`

  Network performance can be monitored with Remote Procedure Call (RPC) statistics. The **`mmdiag`** command with the **`--rpc`** parameter can be used to query RPC statistics. Statistics, such as channel wait time, Receive time TCP, Latency TCP, and Latency verbs, can be useful in diagnosing performance issues, particularly if more details about the timing of RPCs are required.

  For more information about the output of **`mmdiag --rpc`** command, see IBM Knowledge Center.

  In general, this command can be one way of finding out the latency of past RPCs that were exchanged.

Parsing the differences between the “Channel Wait” time (generally and indication of the time spent waiting on access to the TCP socket or RDMA QP), Send time (there are separate statistics to track time in the RDMA and TCP send paths), and Receive times (again, separate statistics to track time in the RDMA and TCP paths).

- `mmdiag --waiters`

  Displays **`mmfsd`** threads that are waiting for events. This information can be helpful in troubleshooting deadlocks and performance problems. For each thread, the thread name, wait time in seconds, and wait reason are typically shown. Only non-idle threads that are waiting for some event to occur are displayed. If file system access seems slow, or IBM Spectrum Scale seems to be hanging, this can be a good place to start debugging.

  Waiters are operations that are taking longer than some threshold, the reporting threshold is hardcoded and is different for each type of operation. Some waiters are normal and indicate a healthy system, and some can provide information about where a problem lies. Many times, waiters are related to a complicated set of interactions between nodes; therefore, it is best to review the waiters across the entire cluster. To run **`mmdiag waiters`** on all the nodes in the cluster at the same time, the **`mmlsnode`** command can be used:

  ```
  mmlsnode -L -N waiters | sort -n -k 3
  ```

  Sorting the data in this manner allows you to easily see the longest waiters as they appear at the end of the output. Often, the longest waiters give the most insight into the root cause of the problem, but this is not always the case.

  There are times where the output of **`mmdiag --waiters`** can help toward narrowing if a problem exists with the network. In some cases, automated monitoring of waiters through some tool or script can help you find issues in your cluster more quickly.

  In some cases, client I/O waiters (that is waiters pending on an I/O to be performed on a remote NSD server) can suggest network problems if the local-IO wait times on the NSD server nodes are low. However, the delay from the client to the server can be occurring in many places, including:

  – Delays in getting access to queue a message to the socket (“waiting for exclusive use of connection” waiters)

  – Delays that are waiting for the message to be copied from the kernel socket buffer, traverse the network, get received on the remote side, and passed to a IBM Spectrum Scale thread polling on the remote socket

  – Delays in the **`mmfsd`** receiver thread path, receiving the message, and getting a worker thread that is dispatched to handle the message

  – Any delay in handling the request on the worker, turning around the reply, and having it received back on the client.

  Included in this category of waiters is a server waiter “getData” that means the server is waiting to get/receive data from the client by way of the network.

  Examples of text found in Client-IO related waiters:

  – “waiting for exclusive use of connection”
  – “for getData on”
  – “for NSD I/O completion”

  The “waiting for exclusive use of connection” is often thought to be an indicator of network issues but it must be carefully interpreted. These waiters indicate delays that are caused by threads that are waiting to gain send privilege (really a lock) to a socket. If threads wait for a long time to send data in this manner, this indicates that the network might not be providing enough bandwidth for optimal performance.

Such waiters can be caused by physical network problems (for example, bad switch cables leading to packet loss in the network), or a configuration issue, such as a misconfiguration of the **`maxMBpS mmchconfig`** variable, which causes IBM Spectrum Scale to try to do too much prefetching, which sends more data than the network can deliver.

4.4 Verifying network with IBM Spectrum Scale mmnetverify command

The **`mmnetverify`** command is highly recommended to be used to verify the network configuration before creating an IBM Spectrum Scale cluster or before adding nodes to an IBM Spectrum Scale Cluster. The command requires that all the nodes under network verification have IBM Spectrum Scale software installed and all the nodes can remote shell to other nodes without the use of a password. The command also can be run to analyze specific network and configuration issues, including the following examples:

- Basic network connectivity issues
- Time synchronization
- Remote shell execution
- Name resolution
- Bandwidth between nodes
- Packet drops between nodes

The **`mmnetverify`** command features the following usage syntax and format:

```
mmnetverify [Operation[ Operation...]] [-N {Node[,Node...] | all}]
            [--target-nodes {Node[,Node...] | all}]
            [--configuration-file File] [--log-file File]
            [--verbose | -Y] [--min-bandwidth Number]
            [--max-threads Number] [--ces-override]
```

For more information about these parameters, see IBM Knowledge Center.

The **`mmnetverify`** command features the following main modes of operation:

- Configuration file (pre-cluster creation)

  This mode is used to verify the network correctness of a group of nodes before the IBM Spectrum Scale cluster is configured. This mode can also be used when you do not want to use the current IBM Spectrum Scale configuration values.

  The **`-configuration-file`** option is used to specify the path of the file. This file is used to describe the nodes of the cluster and the corresponding configuration values. The format of the file is shown in Example 4-4 on page 54. Only the node parameter must be specified (remaining parameters revert to default values if not specified). At least one node entry for the local node must be included in the configuration file.

- Existing cluster

  This mode is used to verify the correctness of an IBM Spectrum Scale cluster by using the information that is obtained from the cluster configuration file `/var/mmfs/gen/mmsdrfs`.

The **`mmnetverify`** command uses the concept of node roles, specifically local node and target node

Local node is the node from which one or more network verification checks that are done by **`mmnetverify`** are started. Checks can be started from one or more nodes. Local nodes are specified with the **`-N`** option with a comma separate list of nodes.

The default is to run on only the node where the **mmnetverify** command is run. If more than one node is specified with **-N** options, the tests run in parallel from these nodes.

Target node is the node against which the test is run. Target nodes are specified with the **-target-nodes (-T)** option as a comma-separated list of node names. The default is to run the checks against all the nodes in the cluster or those nodes that are defined in the configuration file.

Local and target nodes also support node classes.

Operations

Operations are the checks that are done by the **mmnetverify** command and they can directed against the local node, a specific set of nodes, or all the nodes in the cluster.

Local operations

These operations are run only on the local node and do not involve any target node. The only local operation that is supported is "Interface", which checks whether the IP address for the node's daemon and admin interfaces are enabled on the local node.

Node operations

These operations are started from the local node (or the nodes that are specified by **-N**) and always involve one or more target nodes in the cluster. The configuration parameters, such as node names, remote shell, and copy commands that are defined in the cluster configuration file **mmsdrfs** are used for the various checks, which are described next.

Cluster operations

These operations are started from the local node and all the other nodes in the cluster.

For more information about the supported node operations, see IBM Knowledge Center.

Examples of analyzing configuration problems with mmnetverify command

One of the uses of the **mmnetverify** command is to check the correctness of a group of nodes before configuring the IBM Spectrum Scale cluster. The necessary parameters are provided by way of a configuration file. This mode is also used to specify customer parameters different from those defined in the configuration file.

Run the **mmnetverify** command as shown in Example 4-4 to check whether all of the nodes that you want to build a cluster out of have everything set up correctly regarding connectivity, ports, and time synchronization (the output is trimmed for brevity).

Example 4-4 mmnetverify command example

```
$mmnetverify --configuration-file mm.config connectivity port time -N all
$ cat mm.config
node dfnode1 dfnode2
node dfnode2
node dfnode3
# mmnetverify --configuration-file mm.config connectivity port time -n all
Checking on dfnode1: resolution ping shell copy time daemon-port
sdrserv-port tsccmd-port
===============================================
Checking communication with node dfnode1.
Operation resolution: Success.
Operation ping: Success.
Operation shell: Success.
Operation copy: Success.
. . . . . .
```

Example 4-5 shows checking whether the port connectivity is set up correctly before a cluster is configured. We blocked port 1191 on dfnode2 and the command reports the issue as expected (the output is trimmed for brevity).

Example 4-5 mmnetverify checks if the port connectivity is all setup correctly

```
#mmnetverify --configuration-file mm.config port  -N all
Checking on dfnode: daemon-port sdrserv-port tsccmd-port,
==========================================================
Checking communication with node dfnode1.
Operation daemon-port: Success.
Operation sdrserv-port: Success.
. . . . .
Checking on dfnode2: daemon-port sdrserv-port tsccmd-port,
==========================================================
Checking communication with node dfnode1.
Can not run client on node dfnodel - remote shell timeout.
Operation daemon-port: Fail.
Can not run client on node dfnode2 - remote shell timeout.
Operation daemon-port: Fail.
Can not run client on node dfnode2 - remote shell timeout.
Operation sdrserv-port: Fail.
Checking communication with node dfnode3.
Can not run client on node dfnode3 - remote shell timeout.
Operation daemon-port: Fail.
Can not run client on node dfnode3 - remote shell timeout.
Operation sdrserv-port: Fail.
Checking on dfnode3: daemon-port sdrserv-port tsccmd-port
==========================================================
Checking communication with node dfnode1.
Operation daemon-port: Success.
Issues Found:
Remote shell command from dfnode2 to dfnodel failed.
Remote shell command from dfnode2 to dfnode2 failed.
Remote shell command from dfnode2 to dfnode3 failed.
```

Examples of analyzing network issues with the mmnetverify command on a cluster

Consider the following examples in which network issues are analyzed by using the **mmnetverify** command on a cluster:

- The **mmnetverify** command in Example 4-6 checks basic network connectivity problems between a node and other node of the cluster.

  Example 4-6 mmnetverify command checking basic network connectivity problems

  ```
  @dfnode1 ] # mmnetverify ping -N dfnode2 -T dfnode3
  @dfnode1 ] # mmnetverify ping -N dfnode2 -1 dfnode3
  Checking on dfnode2: ping
  =========================
  ```

```
checking communication with node dfnode3.
Operation ping: Success.
No issues found.
```

- The **mmnetverify** command in Example 4-7 checks network and daemon port connectivity problems between the local node and all others nodes of the cluster (the output is trimmed for brevity).

Example 4-7 mmnetverify command checking network and daemon port connectivity problems

```
$mmnetverify connectivity port
# _mmnetverify connectivity port
Checking communication with node dfnode3.
Operation resolution: Success.
Operation ping: Success.
Operation shell: Success.
. . . . . . . . .
No issues found.
```

- The **mmnetverify** command in Example 4-8 checks for problems with network connectivity between each node of the cluster (the output is trimmed for brevity).

Example 4-8 mmnetverify checking network connectivity problems between nodes of the cluster

```
#mmnetverify connectivity -N all -T all
 Checking on dfnode3: resolution ping shell copy
===============================================
Checking communication with node dfnode3.
Operation resolution: Success.
Operation ping: Success.
Operation shell: Success.
. . . . . . .
Checking on dfnode1: resolution ping shell copy
===============================================
Checking communication with node dfnode3.
Operation resolution: Success.
Operation ping: Success.
Checking communication with node dfnode1.
. . . . .
Checking on dfnode2: resolution ping shell copy
===============================================
Checking communication with node dfnode3.
Operation resolution: Success.
Operation ping: Success.
. . . . .
No issues found.
```

- The **mmnetverify** command in Example 4-9 checks problems with large data transfer between two nodes in the cluster.

Example 4-9 mmnetverify command checking problems with large data transfer

```
# mmnetverify data-large -N dfnode2 -T dfnode3 --verbose
Checking on dfnode2: data-large
===============================
Checking communication with node dfnode3.
  Checking network communication with node dfnode3.
```

```
    dfnode3: connecting to node dfnode2.
    dfnode3: exchanged 256. OM bytes with dfnode2.
     Write size: 16. OM bytes.
    Network statistics for dfnode3 during data exchange:
      packets sent: 5646
      packets received: 7920
    Network Traffic between dfnode3 and dfnode2 port 59426 ok.
Operation data-large: Success.

No issues found.
```

- The **mmnetverify** command in Example 4-10 checks the throughput from all the cluster nodes to the local node on which the command is run and reports if the throughput is below the value that is specified in **-min-bandwidth.**

Example 4-10 mmnetverify command checking throughput from cluster nodes to local node

```
Checking cluster communications.
 Checking network bandwidth with all remote nodes.
   dfnode3: connecting to node dfnode1.
    Transfer bandwidth: 746272799
Bandwidth with dfnode3: 746.273M bits per second.
   dfnode1: connecting to node dfnode1.
    Transfer bandwidth: 15161593143
Bandwidth with dfnode1: 15.162G bits per second.
   dfnode2: connecting to node dfnode1.
     Transfer bandwidth: 759327393
Bandwidth with dfnode2: 759.327M bits per second.
Operation bandwidth-cluster: Fail.

Issues Found:
Bandwidth between dfnode1 and dfnode3 below specified minimum.
Bandwidth between dfnode1 and dfnode2 below specified minimum.
```

- The **mmnetverify** command in Example 4-11 checks problems with UDP datagram packet loss between local node and all other nodes of the cluster.

Example 4-11 mmnetverify command checking problems with UDP datagram packet loss

```
# mmnetverify flood-cluster --verbose
Checking cluster communications.
Checking datagram packet loss with all remote nodes.
dfnode3: sending datagrams to node dfnode1.
Received 511,414 of 1,000,000 packets sent from dfnode3 (49% dropped).
Received 51% of packets from dfnode3.
dfnode1: sending datagrams to node dfnode1.
Received 533,171 of 1,000,000 packets sent from dfnode1 (47% dropped).
Received 53% of packets from dfnode1.
dfnode2: sending datagrams to node dfnode1.
Received 397,576 of 1,000,000 packets sent from dfnode2 (61% dropped).
Received 39% of packets from dfnode2.
Operation flood-cluster: Success.
```

- In the **mmnetverify** example that is shown in Example 4-12, we block port `1191` on node `dfnode2` and run a port check. As expected, the checks for `daemon-port` and `sdrserv-port` fails for `dfnode2` (the output is trimmed for brevity).

Example 4-12 mmnetverify command blocked port 1191 test

```
#mmnetverify connectivity port -T dfnode1
Checking communication with node dfnode1
Operation resolution: Success.
Operation ping: Success.
Operation shell: Success.

No issues found
# iptables -A INPUT - p tcp -dport 1191 -j DROP
# mmnetverify connectivity port -T dfnode1

Operation resolution: Success.
Operation ping: Success.
Operation daemon-port: Fail.
Operation sdrserv-port: Fail.

Issues Found:
Remote shell command from dfnode2 to dfnode1 failed.
```

Note: The output is trimmed for brevity.

- The **mmnetverify** command in Example 4-13 checks the CTDB and all the object services ports connectivity from all nodes to the CES nodes in the cluster.

Example 4-13 mmnetverify command checking CTBDB and all object services ports

```
dfnode1 checking communication with node dfnode1.
Operation protocol-ctdb: Success.
Operation protocol-object: Success.

dfnode1 checking communication with node dfnode2.
Operation protocol-ctdb: Success.
Operation protocol-object: Success.

dfnode1 checking communication with node dfnode3.
Operation protocol-ctdb: Skipped.
Operation protocol-object: Skipped.
No issues found.
```

- The **mmnetverify** command in Example 4-14 checks the overall cluster bandwidth from all the target nodes to the local node.

Example 4-14 mmnetverify command checking the overall cluster bandwidth from target nodes

```
#mmnetverify gnr-bandwidth

dfnode1 checking cluster communications.
Bandwidth: 2.447G bits per second.
Operation gnr-bandwidth: Success.
No issues found.
```

Note: Consider the following points about the use of the `mmnetverify` command:

- `mmnetverify` is not a replacement for other network performance tools, such as nsdperf that is available with the IBM Spectrum Scale. The nsdperf tool helps measure network throughput by generating traffic patterns that mimic the way NSD clients and servers communicate. Unlike nsdperf, the `mmnetverify` command does not support running throughput tests over RDMA, many-to-many node throughput tests to provide the overall cluster throughput, or reports node resource stats. Also, all checks are performed within the cluster boundary and it cannot be used to run tests against external entities, such as protocol or object clients or authentication servers.
- If Sudo wrappers are enabled on the cluster, the `mmnetverify` command can be run under sudo on the administration node as the admin user. When sudo wrappers are used, the `-N` option is not supported.

The `mmnetverify` command also helps check if the timestamp in the cluster are in sync and if DNS names of all the nodes are resolvable, but does not specifically check NTP/DNS connectivity.

4.5 Testing network performance with IBM Spectrum Scale nsdperf tool

The IBM Spectrum Scale `nsdperf` is a useful tool that is shipped with IBM Spectrum Scale to assess the cluster network performance and derive troubleshooting insights.

nsdperf overview

By using the `nsdperf` tool, a set of nodes can be defined as clients and servers and a coordinated network test can be run that simulates the IBM Spectrum Scale Network Shared Disk (NSD) protocol traffic. All network communication is done by using the TCP socket connections or RDMA verbs (InfiniBand/iWARP). The `nsdperf` tool is stand-alone and does not use the IBM Spectrum Scale daemon, so it is a good way to test network IO without involving disk I/O. This tool enables effective assessment of the network bandwidth between the NSD client and server nodes pertaining to IBM Spectrum Scale network topology over TCP/IP and over the RDMA network.

When IBM Spectrum Scale software is installed on a node, the `nsdperf` source is installed in the `/usr/lpp/mmfs/samples/net` directory.

nsdperf versus mmnetverify

The `mmnetverify` can be used to assess the network health and verify common network issues in a IBM Spectrum Scale cluster setup. However, the `mmnetverify` tool cannot be used to assess the aggregate parallel network bandwidth between multiple client and server nodes. Also, the tool does not yet support network bandwidth assessment that uses the RDMA protocol. The `nsdperf` and `mmnetverify` tools can be considered as complimenting each other.

nsdperf versus network readiness tool

A new tool, known as the *network readiness tool* was developed to check network across multiple nodes and compare the results against IBM Spectrum Scale Key Performance Indicators (KPI).

This tool attempts to hide much of the complexity of running network measurement tools, such as `nsdperf`, and presents the results in an easy to interpret way. For more information about this open source tool and to download it, see this web page.

nsdperf versus iperf

Network performance programs, such as `iperf3`, are good at measuring throughput between a pair of nodes. However, to use these programs on many cluster nodes requires considerable effort to coordinate start and to gather results from all nodes.

Also, the traffic pattern with many point-to-point streams can give much different results from the IBM Spectrum Scale NSD pattern of clients sending messages round-robin to the servers. Therefore, if iperf3 is producing good throughput numbers but IBM Spectrum Scale file I/O is slow, the problem might still be with the network rather than with IBM Spectrum Scale. The **`nsdperf`** tool can be used for effective network performance assessment pertaining to IBM Spectrum Scale network topology.

When to use nsdperf tool

It is highly recommended to perform cluster network performance assessment by using the **`nsdperf`** tool before the IBM Spectrum Scale cluster deployment to ensure that the underlying network meets the expected performance requirements. Furthermore, if production performance or stability issues occur, it is recommended to quiesce the file system I/O (when permissible) and verify that the underlying network performance is optimal by using the **`nsdperf`** tool.

Complimentary tools

To complement the **`nsdperf`** tool (to aid with IBM Spectrum Scale cluster network performance assessment), the IBM Spectrum Scale **`gpfsperf`** benchmark can be used to measure the end-to-end file system performance (from a IBM Spectrum Scale node) for several common file access patterns. The **`gpfsperf`** benchmark can be run on single node and across multiple nodes.

Two methods are available to achieve parallelism in the **`gpfsperf`** program. More than one instance of the program can be run on multiple nodes by using Message Passing Interface (MPI) to synchronize their execution, or a single instance of the program can run several threads in parallel on a single node. These two techniques can also be combined.

When IBM Spectrum Scale software is installed on a node, the **`gpfsperf`** source is installed in the `/usr/lpp/mmfs/samples/perf` directory.

For more information about building and running **`gpfsperf`** benchmark, see the readme file in the `/usr/lpp/mmfs/samples/perf` directory.

Building nsdperf

For more information about building **`nsdperf`**, see the readme file in the `/usr/lpp/mmfs/samples/net` directory. This section provides a high-level build procedure.

To build nsdperf on GNU/Linux or a Windows system that is running Cygwin/MinGW, run the following command:

```
g++ -O2 -o nsdperf -lpthread -lrt nsdperf.C
```

To build nsdperf with RDMA support (GNU/Linux only), run the following command:

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

The nsdperf built with RDMA support can be saved with different naming scheme (e.g., using **-ib** suffix) to denote the RDMA capability. The **nsdperf** built with RDMA support can also be used to assess TCP/IP network bandwidth in addition to the RDMA network bandwidth.

Note: Because the **nsdperf** (in server mode) tool must be started across multiple nodes, the **nsdperf** binary must be present in all the participating nodes in the same path or location. Consider the following recommendations:

- Build this tool in single node (of similar CPU architecture; for example, x86_64, ppc64) and copy the **nsdperf** binary to a global shared name space (accessible by way of NFS or IBM Spectrum Scale) such that **nsdperf** is accessible from common path.
- Alternatively, the **nsdperf** binary can be built on all the nodes in the `/usr/lpp/mmfs/samples/net` directory by using parallel shell, such as **mmdsh** (for example, **mmdsh -N** all "`cd /usr/lpp/mmfs/samples/net; g++ -O2 -DRDMA -o nsdperf-ib -lpthread -lrt -libverbs -lrdmacm nsdperf.C`").

nsdperf Usage

The **nsdperf** command line options are shown in Example 4-15 and also described in the README in the `/usr/lpp/mmfs/samples/net` directory (on node with IBM Spectrum Scale installed).

Example 4-15 nsdperf command syntax and options

```
nsdperf-ib [-d] [-h] [-i FNAME] [-p PORT] [-r RDMAPORTS] [-t NRCV] [-s] [-w
NWORKERS] [-6] [CMD…]

Options:
-d Include debug output
-h Print help message
-i FNAME Read commands from file FNAME
-p PORT TCP port to use (default 6668)
-r RDMAPORTS RDMA devices and ports to use (default is all active ports)
-t NRCV Number of receiver threads (default nCPUs, min 2)
-s Act as a server
-w NWORKERS Number of message worker threads (default 32)
-6 Use IPv6 rather than IPv4
```

Generally, the most often used **nsdperf** command **-line** option is **-s**, which is used to start **nsdperf** in server mode. The **nsdperf** in server mode must be run on all of the nodes of cluster (that must be involved in the **nsdperf** testing where network bandwidth between the NSD client and server must be assessed); for example:

```
mmdsh -N <participating_nodes> '<complete_path_to>nsdperf -s </dev/null > /dev/null 2>&1 &'
```

After the **nsdperf** servers are running, the network bandwidth assessment between NSD client and servers can be performed by running **nsdperf**, without **-s**, from an administrative node (for example, login node, gateway node, or any cluster node that permits interactive job execution), and entering **nsdperf** commands:

```
<complete_path_to>nsdperf
```

The **test** command sends a message to all the client nodes to begin write and read network performance testing to the server nodes. The size of the messages can be specified by using the **nsdperf buffsize** parameter. It is suggested to start with **nsdperf buffSize NBYTES** equal to the IBM Spectrum Scale file system block size to assess the network bandwidth capability.

When sequential I/O is performed on the IBM Spectrum Scale file system, the NSD clients transmit I/O sizes in units of file system block size to the NSD servers.

Throughput numbers are reported in MBps, where MB is 1,000,000 bytes. The CPU busy time during the test period is also reported (currently only supported on Linux and AIX systems), as described in the **nsdperf** README in the `/usr/lpp/mmfs/samples/net` directory (on node with IBM Spectrum Scale installed). The numbers that are reported are the average percentage of non-idle time for all client nodes, and average for all server nodes.

The following **nsdperf** test types are available:

- write

  Clients write round-robin to all servers. Each client tester thread is in a loop, writing a data buffer to one server, waiting for a reply, and then moving on to the next server.

- read

  Clients read round-robin from all servers. Each client thread sends a request to a server, waits for the data buffer, and then moves on to the next server.

- nwrite

  This test is the same as the write test, except that it uses a IBM Spectrum Scale NSD style of writing, with a four-way handshake. The client tester thread first sends a 16-byte NSD write request to the server. The server receives the request and sends back a read request for the data. The client replies to this with a data buffer. When the server receives the data, it replies to the original NSD write request, and the client gets this information and moves on to the next server.

- swrite

  Each client tester thread writes repeatedly to a single server, rather than sending data round-robin to all servers. To get useful results, the **threads** command are used to make the number of tester threads be an even multiple of the number of server nodes.

- sread

  Each tester thread reads from only one server.

- rw

  This test is bidirectional in which half of the client tester threads run the read test and half of threads run the write test.

At the minimum, the network bandwidth assessment should be performed by using write, read, and nwrite tests. The nwrite test is pertinent test when IBM Spectrum Scale cluster is deployed over the InfiniBand network and the IBM Spectrum Scale **verbsRdma** configuration parameter is enabled.

Special considerations for IBM Spectrum Scale clusters with InfiniBand

The **nsdperf** command-line option **-r** must be provided with value same as the IBM Spectrum Scale **verbsPorts** parameter (**mmlsconfig | grep verbsPorts**). The format of the **RDMAPORTS** argument of the **-r** option is a comma or space-separated list of device names and port numbers that are separated by colon or slash (for example, **mlx5_0/1**, **mlx5_1/1**).

When multiple ports are specified, RDMA connections are established for each port and outbound messages are sent round-robin through the connections. If a port number is not specified, all active ports on the device are used.

After the **nsdperf** servers are running, on an administrative node (for example, login node, gateway node, or any cluster node permitting interactive job execution) run **nsdperf** with the same **-r** value used for **nsdperf** in server mode (**-s**).

For example, if IBM Spectrum Scale **verbsPorts** is set to **mlx5_0/1**, **nsdperf** in server mode (**-s**) must include **RDMAPORTS** (**-r**) set to **mlx5_0/1**), as shown in following example:

```
mmdsh -N <participating_nodes> '<complete_path_to_>nsdperf-ib -s -r mlx5_0/1
</dev/null > /dev/null 2>&1 &'
```

The **nsdperf** administrative command (without **-s** option) must include **RDMAPORTS (-r)** that is set to **mlx5_0/1**), as shown in the following example:

```
<complete_path_to>nsdperf-ib -r mlx5_0/1
```

Various nsdperf tests

When troubleshooting network issues, testing the following **nsdperf** combinations helps with determining the issue:

- Single client and single server (between IBM Spectrum Scale clients, IBM Spectrum Scale client, and NSD server)
- Single client and multiple Servers (IBM Spectrum Scale client and multiple NSD servers)
- Multiple clients and multiple servers (Multiple clients and multiple NSD servers)

For more information about examples of **nsdperf** tests, see Appendix B, “nsdperf command examples” on page 87.

4.6 IBM Spectrum Scale mmuserauth check

When IBM Spectrum Scale protocol nodes are configured, configuration with external authentication server becomes a prerequisite. Suitable configuration and network connectivity with external authentication server is a requirement for graceful operation of the protocol nodes. The **mmuserauth check** command verifies the authentication method configuration details for file and object access protocols. It also features the **server-reachability** option that helps validate the connectivity to the configured authentication servers. It also supports corrections to the configuration details about the erroneously configured protocol nodes, as shown in the following example:

```
mmuserauth service check [--data-access-method {file|object|all}] [-r|--rectify]
            [-N|--nodes {node-list|cesNodes}][--server-reachability]
```

For more information, see IBM Knowledge Center.

Example 4-16 shows the **mmuserauth service check** command.

Example 4-16 mmuserauth service check command

```
mmuserauth service check --server-reachability
Userauth object check on node: vmnode2
       Checking keystone.conf: OK
       Checking wsgi-keystone.conf: OK
       Checking /etc/keystone/ssl/certs/signing_cert.pem: OK
       Checking /etc/keystone/ssl/private/signing_key.pem: OK
       Checking /etc/keystone/ssl/certs/signing_cacert.pem: OK

LDAP servers status
       LDAP server 9.118.37.234 : OK
Service 'httpd' status: OK
```

In Example 4-16 on page 63, the protocol authentication is configured for object protocol with LDAP being the authentication server. The command successfully checks the LDAP connectivity with the protocol nodes.

4.7 Using IBM Spectrum Scale log messages to debug network-related issues

When looking at a problem from a network perspective, it can be helpful to determine if there is some type of socket-level communication problem between IBM Spectrum Scale daemons. This issue can point to a problem below IBM Spectrum Scale (for example, sockets with high TCP back off values or long TCP retransmission timeout values). For more information, see "Debugging expels" on page 72.

Example 4-17 shows sample log entries that show network issues.

Example 4-17 Sample log entries for network issues

```
hoststrhep03g:2018-12-10_10:16:16.565+0000:2018-12-10_10:16:16.565+0000: [N]
Request sent to XX.XX.XXX156 (hoststrmp01g) to expel XX.XX.XXX.75 (hoststrdbp12g)
from cluster XXX.gpfscluster.com
hoststrdbp31g:2018-12-10_10:47:43.056+0000:2018-12-10_10:47:43.056+0000: [E] The
TCP connection to IP address XX.XX.XXX.75 hoststrdbp12g <c0n17> (socket 283) state
is unexpected: ca_state=4 unacked=11 rto=25728000
hoststrdbp12g:2018-12-10_10:52:50.512+0000:2018-12-10_10:52:50.512+0000: [N]
Request sent to XX.XX.XXX.156 (hoststrmp01g) to expel XX.XX.XXX.19 (hoststrdbp31g)
from cluster XXX.gpfscluster.com
hoststrdbp27g:2018-12-10_10:53:03.042+0000:2018-12-10_10:53:03.042+0000: [E] The
TCP connection to IP address XX.XX.XXX.75 hoststrdbp12g <c0n17> (socket 257) state
is unexpected: ca_state=4 unacked=5 rto=3520000
hoststrdbp22g:2018-12-10_10:53:03.076+0000:2018-12-10_10:53:03.076+0000: [E] The
TCP connection to IP address XX.XX.XXX.75 hoststrdbp12g <c0n17> (socket 258) state
is unexpected: ca_state=4 unacked=6 rto=3424000
hoststrdbp27g:2018-12-10_10:58:43.204+0000:2018-12-10_10:58:43.204+0000: [N]
Request sent to XX.XX.XXX.156 (hoststrmp01g) to expel XX.XX.XXX.75 (hoststrdbp12g)
from cluster XXX.gpfscluster.com
hoststrdbp22g:2018-12-10_10:58:43.983+0000:2018-12-10_10:58:43.983+0000: [N]
Request sent to XX.XX.XXX.156 (hoststrmp01g) to expel XX.XX.XXX.75 (hoststrdbp12g)
from cluster XXX.gpfscluster.com
hoststrdbp02g:2018-12-10_10:58:53.307+0000:2018-12-10_10:58:53.307+0000: [E] The
TCP connection to IP address XX.XX.XXX.75 hoststrdbp12g <c0n17> (socket 283) state
is unexpected: ca_state=4 unacked=4 rto=3456000
hoststrdbp02g:2018-12-10_11:04:25.809+0000:2018-12-10_11:04:25.809+0000: [N]
Request sent to XX.XX.XXX.156 (hoststrmp01g) to expel XX.XX.XXX.75 (hoststrdbp12g)
from cluster XXX.gpfscluster.com
hoststrmp05g:2018-12-10_11:14:11.203+0000:2018-12-10_11:14:11.203+0000: [E] The
TCP connection to IP address XX.XX.XXX.75 hoststrdbp12g <c0n17> (socket 90) state
is unexpected: ca_state=4 unacked=9 rto=25728000
```

4.8 Node expels because of network issues

In IBM Spectrum Scale environments, two primary symptoms are observed because if network issues: node expels and performance problems. The following sections provide more information about expels.

Expels

An expel refers the case in which a node is suspended from accessing an IBM Spectrum Scale cluster's file systems. An expelled node is sometimes referred to as having “lost cluster membership” or “lost quorum” (this use is a different use of the term “quorum” than was used in “cluster node quorum” or “ file system descriptor quorum“).

Effect of expel

After a node is expelled from a cluster, the expelled node can no longer submit I/O requests, its tokens and locks are released, and its IBM Spectrum Scale file systems are unmounted. Applications that are running on the expelled node can fail after receiving an input/output error (EIO) or ENOENT (no such file or directory) response to an I/O request that is made to an unmounted file system. After an unmount occurs, an application might receive a terminating SIGBUS signal if its binary is stored on the file system that becomes unmounted.

Why does IBM Spectrum Scale need to expel nodes?

To ensure data integrity and good performance, while preventing hangs and deadlocks, IBM Spectrum Scale requires reliable communication between all nodes by using a file system.

A simple example illustrates this requirement. Consider a case in which a Node X, opens a file, gets a byte range write lock to modify a region in the file, and then, over-commits memory such that it then becomes unresponsive.

After the memory is over committed, if any other node must access the region of the file that Node X locked, a request can be sent to Node X to revoke its byte range lock. Without an expel mechanism to force the freeing up of resources that are held by unresponsive nodes, all other nodes in the cluster might experience a hang if Node X (because its memory was over-committed) fails to process any such revoke requests.

To preserve the shared file system data integrity, IBM Spectrum Scale uses a disk lease mechanism to prevent nodes from starting unauthorized I/O operations in the cluster. It also uses an expel process to remove non-responsive nodes from the cluster.

Next, we describe the IBM Spectrum Scale lease process and configuration settings.

Recovery from expels

After a node is expelled from a IBM Spectrum Scale cluster and all recovery protocols are run, the cluster manager allows the expelled node to rejoin the cluster by issuing a probe request to the cluster manager.

Note: When a node is specifically disabled by running the **`mmexpelnode`** command, it does not attempt to rejoin the cluster. The **`mmexpelnode`** command is one way of expelling a node that is in a bad state; for example, if memory is over-committed on the affected node, but you want to avoid restarting the node.

Expels can be caused by many issues, including the following examples:

- Network connectivity problems, which are not always a hardware layer issue; for example, sysctl tuning, such as preloading of the ARP tables, which should be done on large InfiniBand clusters (such as 1000+ node clusters).
- Resources issues on the cluster (memory over-commitment on the failing or expelled node is the most common resource issue).
- Issues in IBM Spectrum Scale code. Of these issues, the most significant is described in 4.2.2, "Monitoring Network Performance Using the GUI" on page 47 by way of an RPC prioritization change and a second RPC prioritization change was added in IBM Spectrum Scale 5.0.2).
- Other software causing IBM Spectrum Scale threads to be blocked (check for this issue by reviewing the console log from the time a failure and look for "stuck CPU" warnings or threads that are blocked for more than 120 seconds).

Types of expels

IBM Spectrum Scale features the following types of expels:

- Disk lease timeout related expels: These expels are started by the cluster manager after a node fails to renew its disk lease back to the cluster manager.
- RPC timeout related node requested expel: When a node makes an RPC request to another node, the source of the request might later ask the target to acknowledge that the RPC message is still pending. If the target does not acknowledge back to the source that the request is pending, the source asks the cluster manager to expel the target node, and then the cluster manager expels the source or target node.
- Expels that are requested by running the **`mmexpelnode`** command: An IBM Spectrum Scale admin might cause a node to be expelled by running the **`mmexpelnode`** command. This command was used in some solutions to automate recovery; for example, some IBM DB2® configurations that use RSCT to determine when nodes become unavailable.

Disk lease timeout-related expels (cluster manager initiated expels)

In each cluster, the cluster manager manages *disk leases*, and disk leases are needed to submit I/O requests to a specific IBM Spectrum Scale cluster.

Each node (regardless of its role) must renew its lease with the cluster manager node at regular intervals (leases are renewed every 35 seconds by default) or the node no longer can submit I/O requests to the cluster. Shortly after lease expiration occurs, the cluster manager might expel the node that has an expired lease. Therefore, failure to renew leases in a timely manner also results in expels (see Figure 4-1).

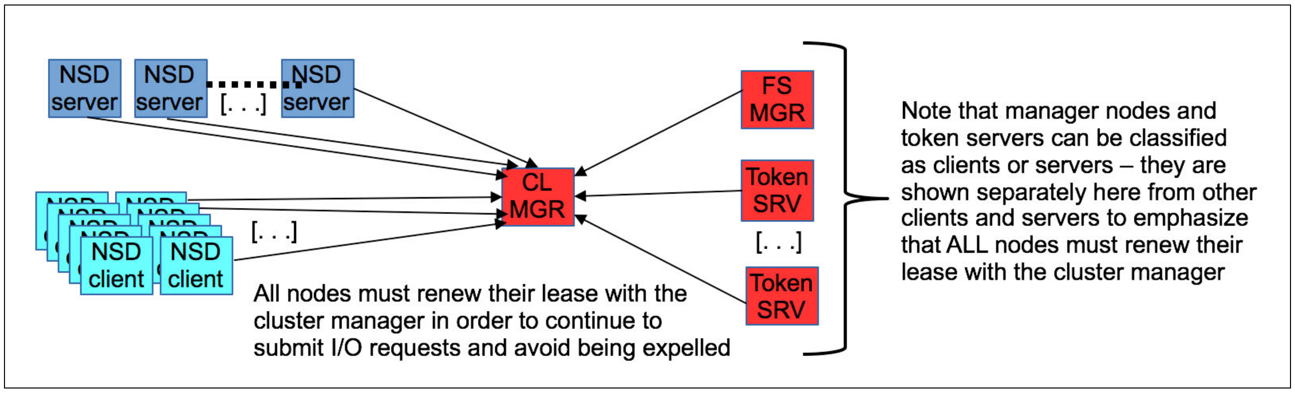

NSD server
NSD server
[. . .]
NSD server
NSD client
NSD client
[. . .]
CL MGR
FS MGR
Token SRV
[. . .]
Token SRV
All nodes must renew their lease with the cluster manager in order to continue to submit I/O requests and avoid being expelled
Note that manager nodes and token servers can be classified as clients or servers – they are shown separately here from other clients and servers to emphasize that ALL nodes must renew their lease with the cluster manager

Figure 4-1 All nodes must renew their lease with cluster manager (CL)

Flow (and tuning) disk lease timeout-related expels

Figure 4-2 shows the high-level lease timeout flow when disk leasing is in effect. In this publication, we focus on the disk leasing case and review the effect of configuration parameter tuning.

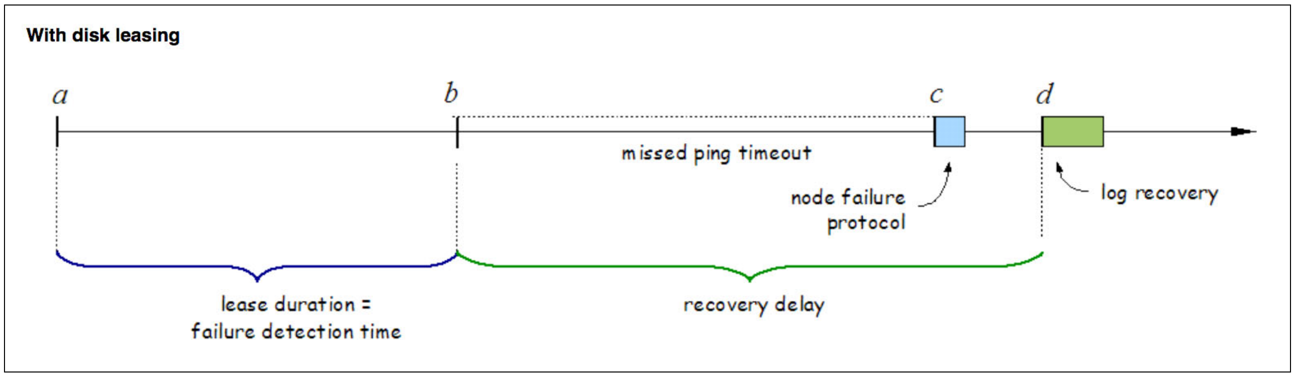


Figure 4-2 High-level lease timeout flow when disk leasing is in effect

The length of a non-quorum node's lease (in seconds) is defined by the configuration options **`leaseDuration`** and **`failureDetectionTime`**. Based on large cluster experience, we recommend that if required, tune the **`failureDetectionTime`** and not **`leaseDuration`** setting **`failureDetectionTime`** also changes **`leaseDuration`**).

A node is not expelled until after its lease expires (lease duration=failure detection time in seconds) and then the "missed ping timeout" window completes. In the "missed ping timeout" window, the cluster manager sends ICMP datagrams (pings) to the node with the expired lease. The length of the "missed ping timeout" window is dependent the node's response to the cluster manager's pings as described next.

If a node fails to respond to pings from the cluster manager, the length of the "missed ping timeout" window is defined by the **`mmfsd`** computed value **`missedPingTimeout`**, which defaults to 30 seconds. IBM Spectrum Scale sets the value of **`missedPingTimeout`** based on the value of IBM Spectrum Scale configuration parameters, taking the maximum of **`minMissedPingTimeout`** and **`leaseRecoveryWait`** (but never exceeding **`maxMissedPingTimeout`**).

If a node responds to pings from the cluster manager, the length of the "ping timeout" window is defined by the IBM Spectrum Scale configuration parameter **`totalPingTimeout`**, which defaults to 120 seconds.

When trying to run IBM Spectrum Scale on a network that is less resilient (for example, has a packet loss problem), some customers choose to increase the length of the "missedPingTimeout" window to allow nodes to be specific more time before they are expelled (after their lease expires). Example 4-18 shows one such set of configuration options was tested in the field to try to ride over short windows in which network connectivity might break (these parameters require a restart of IBM Spectrum Scale to take effect).

Example 4-18 Configuration options to try to ride over longer network outages without expel

```
#Increase the minMissedPingTimeout and failureDetectionTime values and ride over
longer network outages without expel
mmchconfig minMissedPingTimeout=60
mmchconfig failureDetectionTime=60
```

Checking IBM Spectrum Scale lease-related configuration values

Example 4-19 shows the relevant lease tuning (**mmchconfig**) configuration values by using **mmdiag** (**mmlsconfig** can also be run to query each of these values).

Example 4-19 relevant lease tuning (mmchconfig) configuration values using mmdiag

```
# /usr/lpp/mmfs/bin/mmdiag --config|grep -e failureDetectionTime -e
leaseDMSTimeout -e leaseDuration -e leaseRecoveryWait -e MissedPingTimeout
failureDetectionTime -1
leaseDMSTimeout -1
leaseDuration -1
leaseRecoveryWait 35
maxMissedPingTimeout 60
minMissedPingTimeout 3
```

Example of disk lease timeout expel reported on the cluster manager

One of the best ways to determine whether a network layer problem is root cause for an expel is to review the low-level socket details that are dumped in the `internaldump` log data that is saved as part of automatic data collection on Linux IBM Spectrum Scale nodes (not on AIX or Windows). Some of this socket-level data also is printed to the logs in later code levels, when IBM Spectrum Scale detects an issue with a socket that has outstanding messages that are associated with it.

The (`mmfs.log`) log file (see Example 4-20) also show results of pings that are sent from the cluster manager to the failing node, before the expel, but we cannot always trust the correct underlying interface is pinged in the case of an 802.3ad xmit layer 3+4 bonded configuration. This issue also is true when the subnets configuration option is used (this problem is fixed in IBM Spectrum Scale 5.0.2, because this level of code pings the correct underlying interface). If the pings do not go through (if “Replies received: 0"), it can indicate a node failure, a reboot, or a network connectivity issue.

Example 4-20 mmfs.log records showing failed ping results from cluster manager to the failing node

```
2018-04-01_18:45:54.994-0400: [E] The TCP connection to IP address XX.XX.2.3
c933f02x03 <c0n0> (socket 67) state is unexpected: ca_state=4 unacked=1
rto=4000000
2018-04-01_18:45:54.994-0400: [I] tscCheckTcpConn: Sending debug data collection
request to node XX.XX.2.3 c933f02x03
2018-04-01_18:46:00.420-0400: [E] Node XX.XX.2.7 (c933f02x07) is being expelled
because of an expired lease. Pings sent: 60. Replies received: 60.
2018-04-01_18:46:04.302-0400: [E] Node XX.XX.2.5 (c933f02x05) is being expelled
because of an expired lease. Pings sent: 60. Replies received: 60.
```

Disk lease-related process: DMS timeouts

Nodes in a IBM Spectrum Scale cluster have a requirement for all I/Os to complete within a threshold of time after disk lease expiration. To preserve the integrity of the file system, IBM Spectrum Scale must ensure that no delayed or unexpected I/Os are issued to the disks while log recovery is occurring. If pending I/Os exist that did not complete after a server node’s lease expires, this issue can trigger IBM Spectrum Scale to panic (crash) the affected server node. This panic is to prevent file system corruption.

When a server node renews its lease, it schedules a timer event **leaseDMSTimeout** seconds after the point at which a lease timeout occurs. When the timer event occurs, if pending local I/Os to the IBM Spectrum Scale cluster file system still exist and the node in question still did not renew its lease, the IBM Spectrum Scale kernel extension starts a panic event that crashes the node. (In this context, *local I/Os* are defined as I/Os to the storage that is managed or owned by the server, but the server no longer has a valid disk lease for this storage.) This panic mechanism is referred to as the *Dead Man Switch* (DMS) timeout.

Disk lease-related process: Cluster manager takeover

The cluster manager can never be expelled from a IBM Spectrum Scale cluster, but it might experience failures that have a similar nature to an expel.

The cluster manager decides to renew its own leases, based on the number of quorum nodes that renew their lease to the cluster manager. For the cluster manager to renew its own lease, it must see that most quorum nodes must renew their leases to the manager; it is through this process that node quorum is verified.

Example 4-21 shows `mmfs.log` entries that are reported on the cluster manager when it does not receive lease renewals from other nodes in the cluster. This issue leads to the cluster manager resigning as a result of “Not enough quorum nodes reachable”.

Example 4-21 also shows a connectivity problem that was created by blocking TCP traffic by way of iptables.

Example 4-21 mmfs.log entries reported by cluster manager when it doesn't receive lease renewals

```
2018-04-01_18:43:54.656-0400: [N] Disk lease period expired 2.600 seconds ago in
cluster gpfs-cluster.c933f02x01. Attempting to reacquire the lease.
2018-04-01_18:44:00.426-0400: [N] Node XX.XX.2.7 (c933f02x07) lease renewal is
overdue. Pinging to check whether it is alive
2018-04-01_18:44:04.307-0400: [N] Node XX.XX.2.5 (c933f02x05) lease renewal is
overdue. Pinging to check whether it is alive
2018-04-01_18:44:04.787-0400: [N] Node XX.XX.2.3 (c933f02x03) lease renewal is
overdue. Pinging to check whether it is alive
2018-04-01_18:45:48.948-0400: [E] Close connection to XX.XX.2.3 c933f02x03 <c0n0>
(Connection reset by peer). Attempting reconnect.
[. . .]
2018-04-01_18:45:49.948-0400: [N] Connecting to XX.XX.2.3 c933f02x03 <c0n0>
2018-04-01_18:45:50.949-0400: [N] Connecting to XX.XX.2.15 c933f02x15 <c0n6>
[. . .]
2018-04-01_18:45:54.994-0400: [E] The TCP connection to IP address XX.XX.2.3
c933f02x03 <c0n0> (socket 67) state is unexpected: ca_state=4 unacked=1
rto=4000000
2018-04-01_18:45:54.994-0400: [I] tscCheckTcpConn: Sending debug data collection
request to node XX.XX.2.3 c933f02x03
2018-04-01_18:46:00.420-0400: [E] Node XX.XX.2.7 (c933f02x07) is being expelled
because of an expired lease. Pings sent: 60. Replies received: 60.
2018-04-01_18:46:04.302-0400: [E] Node XX.XX.2.5 (c933f02x05) is being expelled
because of an expired lease. Pings sent: 60. Replies received: 60.
2018-04-01_18:46:04.302-0400: [E] Not enough quorum nodes reachable: 2.
2018-04-01_18:46:04.302-0400: [E] Lost membership in cluster
gpfs-cluster.c933f02x01. Unmounting file systems.
2018-04-01_18:46:04.408-0400: [N] Close connection to XX.XX.2.3 c933f02x03 <c0n0>
(Node failed)
2018-04-01_18:46:05.954-0400: [I] Calling user exit script mmUnmountFs: event
unmount, Async command /usr/lpp/mmfs/lib/mmsysmon/sendRasEventToMonitor.
```

```
2018-04-01_18:46:05.971-0400: [E] File /tmp/unmount_data_collection cannot run
with error 2: No such file or directory.
2018-04-01_18:46:05.981-0400: [I] Quorum loss. Probing cluster
gpfs-cluster.c933f02x01
```

As shown in Example 4-21 on page 69, if the cluster manager decides to stop renewing its own lease, it no longer responds to lease renewal requests from other nodes and resigns from the cluster, declaring “Lost membership”.

In the case of the cluster manager resigning or failing, the remaining quorum nodes start a process that causes them to elect a new cluster manager because they cannot renew their own leases. The quorum nodes use a shorter disk lease (two-thirds of the lease timeout that is used by other nodes) so they can efficiently validate the existence of a functioning cluster manager and, if needed, more quickly start the process of cluster manager takeover.

RPC timeout-related expels (node requested expels)

Any node can make an RPC request to any other node in the cluster. Requests might timeout (which leads to an expel) in the following circumstances:

- If the node that needs to make a request cannot establish a connection in a timely manner to the node that is the target of the request, the requesting node asks the cluster manager to expel the target node.
- If a connection is established from the requesting node to the target of the request, but the request is not handled in a timely manner, the node making the request will ask the target of the request to acknowledge that the request is pending. If that acknowledgment is not sent back in a timely manner, the requester will ask the cluster manager to expel the target node.

If the node making the RPC request hits one of these two circumstances, it asks the cluster manager to expel the target. Then, the cluster manager expels either requester node, or the target of the request (see Figure 4-3). For more information, see “RPC timeout node requested expels: Which node is expelled?” on page 71.

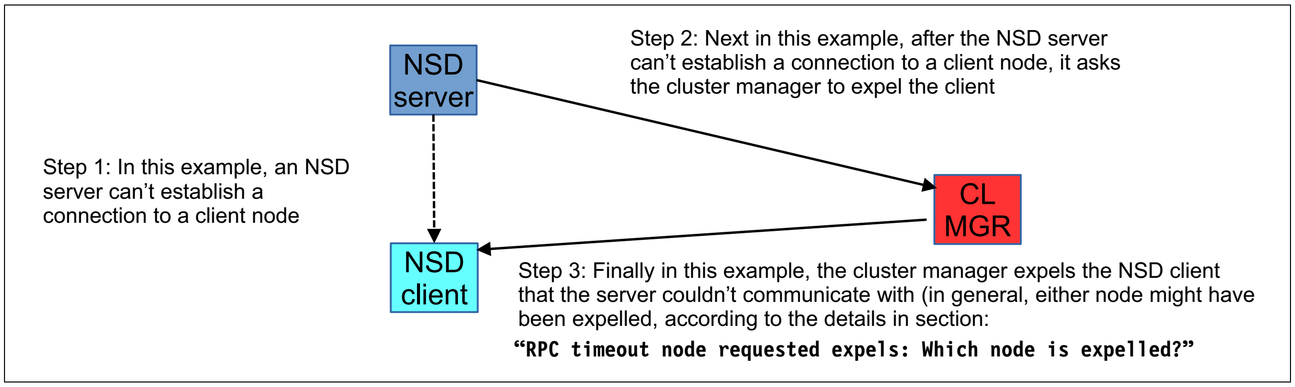

NSD server
Step 2: Next in this example, after the NSD server can’t establish a connection to a client node, it asks the cluster manager to expel the client
Step 1: In this example, an NSD server can’t establish a connection to a client node
CL MGR
NSD client
Step 3: Finally in this example, the cluster manager expels the NSD client that the server couldn’t communicate with (in general, either node might have been expelled, according to the details in section:
“RPC timeout node requested expels: Which node is expelled?”

Figure 4-3 RPC timeout node requested expels process

Every 90 seconds, the **`EEWatchDogThread`** in the IBM Spectrum Scale daemon looks for pending messages that are outstanding to all destinations. For every destination that has pending messages, the IBM Spectrum Scale daemon sends a **`commMsgCheckMessages`** message that attempts to verify that the destination is aware of all pending messages and can acknowledge that these messages are being worked.

The timeout on the **commMsgCheckMessages** is calculated to be the minimum of (10 * leaseDuration) and (300). Because the default value for **leaseDuration** is 35, the timeout typically ends up being 300 seconds on a non-quorum node (quorum nodes use two-thirds the typical lease length). The timeout on **commMsgCheckMessages** includes sending a single retry of the message to allow for timing holes in how the pending messages are viewed on the source and target nodes.

If the target of a specific message did not acknowledge the **commMsgCheckMessage** for a message that reached the timeout value (again, by default this is 300 seconds), the source node requests the cluster manager to expel the target node. The cluster manager, then, needs to decide whether it should expel the source or destination node (for more information about that process, see "RPC timeout node requested expels: Which node is expelled?" on page 71).

Potential shorter timeouts might exist that can lead to RPC timeout node requested expels, in addition to the default timeout of 300 seconds that occurs when communicating with destinations that have connections established.

If a node must make an RPC call to another node to which no established connection exists, a shorter timeout can be hit in the code paths that establish new connections. The following examples of these shorter timeout cases are for TCP connections:

- When a new TCP connection must be established to another host, if ARP resolution is not working and a valid ARP entry is not present for the destination IP, a "NO ROUTE TO HOST" error occurs. This error establishes the connection, by default after 3 seconds of ARP retries (controlled by the applicable **sysctl ucast_solicit** parameter). IBM Spectrum Scale retries the connection, but the RPC can time out in as little as 6 seconds if ARP is not working and the default tuning is in effect.
- If ARP is working, but TCP connections cannot be established, an RPC timeout can, with default TCP tuning, occur in approximately two minutes. The default timeout on establishing a new TCP connection is about one minute. IBM Spectrum Scale retries the TCP connection once, and the TCP connection times-out.

RPC timeout node requested expels: Which node is expelled?

In the case of a node-requested expel, the default policy when deciding which node should be expelled by the cluster manager is controlled by a default set of priority choices that is made by the cluster manager. In terms of which nodes are favored (preferred) to keep active in the cluster, the following defaults are used:

- Quorum nodes are favored over non-quorum nodes
- Local (home file system) cluster nodes are preferred over nodes accessing a remote file system
- Nodes having a management role are favored over non-manager nodes
- Nodes that are managing more file systems are preferred over nodes that are managing fewer file systems
- NSD servers are favored over non-NSD server nodes
- Nodes that were active longer in the cluster are preferred over nodes that were part of the cluster for shorter periods of time.

IBM Spectrum Scale provides a callback mechanism to allow changing these defaults to provide more precise control over which node is expelled on an RPC timeout-related node requested expel (for more information about for how to use this callback, see `/usr/lpp/mmfs/samples/expelnode.sample`).

The callback script can also be used to take action on such an expel, such as calling-out an admin, shutting down services, and generating an alert.

To expel a node in this manner, **`mmexpelnode`** is run with the **`-N`** flag to specify the target nodes for the expel; for example, **`mmexpelnode -N c933f02x27`**.

Expels caused by the mmexpelnode command

A system administrator can cause a node to be expelled by running the **`mmexpelnode`** command. Expels that are forced in this manner have the same impact as an expel that results from the more common lease timeout or RPC timeout node requested expel flows. Running the **`mmexpelnode`** command can be useful when it is wanted to remove a node from the IBM Spectrum Scale cluster without shutting down the node (for example, you might want to temporarily expel a node that has a resource issue, such as over-committed memory).

Warning: Do not use the **`-f`** (**`--is-fenced`**) option to **`mmexpelnode`**, unless you are sure that the node is down or referenced (this option avoids the typical log recovery delay that is associated with an expel). In general, it is recommended to avoid this **`--is-fenced`** option.

To expel a node in this manner, run **`mmexpelnode`** with the **`-N`** flag to specify the target nodes for the expel, as shown in the following example:

```
/usr/lpp/mmfs/bin/mmexpelnode -N c933f02x27
```

To bring a node back into the cluster after it is been in this manner, use the **`-r`** flag, as shown in the following example:

```
/usr/lpp/mmfs/bin/mmexpelnode -r -N c933f02x27
```

Debugging expels

One good approach to debugging an expel is to look for any cases in which IBM Spectrum Scale reported low-level socket details for a connection that was involved in an expel. In general, the latest IBM Spectrum Scale code automatically captures socket-level statistics about the connections it is using in the following cases:

- The IBM Spectrum Scale daemon (**`mmfsd`**) had a pending message on a socket. When it reviewed the low-level socket statistics, it found indications that the TCP connection was in a “bad state”, which causes it to log an entry to `mmfs.log`.

  IBM Spectrum Scale determines a connection to be a “bad state” if either of the following conditions is true:

  – The CA_STATE for the socket is not in 0 (or open) state, which means the state must be disorder, recovery, or loss. For more information about CA_STATE, see *TCP’s Congestion Control Implementation in Linux Kernel*.

  – The RTO for the socket is greater than 10 seconds and unacknowledged messages exist that pending on the socket (unacked > 0).

    Older versions of the IBM Spectrum Scale code had a third criteria in checking for “bad state”, which included checking whether unacked was greater than 8, but that check sometimes called out a socket that was working, so this third check was removed by way of the APAR IJ02566. All IBM Spectrum Scale V5 code has this fix and the 4.2.X code stream picked up this fix in PTF 7 (4.2.3.7 ships APAR IJ02566).

- By default, for expel and “deadlock detected” (that is, “long waiters”) events, IBM Spectrum Scale should automatically collect internal dump files that contain (**`mmfsadm dump all`**) socket state data.

For the case of an expel, automated data collection should collect the required dump data containing socket state data on both sides of the key two nodes involved. That is, the first circumstance in the case of a lease timeout, the cluster manager and node with an expired lease; and in the second circumstance of an RPC timeout node requested expel, both the node making the RPC request and the node that did not respond to the request in time. (A threshold of data exists that is collected, according to how variables, such as **expelDataCollectionDailyLimit**, are set.)

In the internal dump data, jump to the "dump tscomm" section and find the socket details for the other node of interest and check for indications of a problem at the TCP layer.

Of the TCP socket level data that is collected, the following most useful fields can be reviewed when debugging expels or network issues in general:

- rto

  This field represents the TCP retransmission timeout in microseconds. This value shows the time that it takes TCP to issue a non-fast path retransmission on the socket. Higher values here indicate that TCP has completed its back off algorithm after retransmitting several times. The timer typically starts out at approximately 204000 on a low latency network, and higher values can suggest packet loss or other delays (for example, flow control pauses) that indicate a network-related issue outside of the IBM Spectrum Scale code itself.

- retransmits, backoff, lost

  If any of these fields are nonzero, the socket is retransmitting and we expect the RTO on the associated socket also to rise.

- ca_state

  If this field is anything other than "open", the socket experienced a packet loss or an out-of-order packet delivery issue.

Example 4-22 shows a socket connection that has a clear problem (TCP backed off the RTO to 120000000; therefore, even if the network recovers when the socket is in this state, it can take up to 120 seconds for data to start flowing on the socket again).

Example 4-22 Socket connection problem

```
192.168.1.13/1
    state 1 established snd_wscale 7 rcv_wscale 9 rto 120000000 ato 40000
    retransmits 0 probes 0 backoff 11 options: WSCALE
    rtt 3711 rttvar 123 snd_ssthresh 2 snd_cwnd 2 unacked 122
    snd_mss 1460 rcv_mss 973 pmtu 2044 advmss 2004 rcv_ssthresh 31074
    sacked 0 lost 120 retrans 0 fackets 0 reordering 3 ca_state 'loss'
```

The following example is a socket connection that shows an RTO that is above the threshold that **mmfsd** uses for writing a socket "state is unexpected" message. In this example, the IBM Spectrum Scale deamon has a message pending to IP address 192.168.3.4 and the RTO has risen to approximately 27 seconds:

```
2018-08-23_09:28:12.923-0500: [E] The TCP connection to IP address 192.168.3.4
some client <c0n12> (socket 21) state is unexpected: ca_state=1 unacked=3
rto=27008000
```

One of the goals of the approach of looking at TCP/IP socket level statistics is to isolate the problem that is leading to a functional failure, such as an expel. If IBM Spectrum Scale is trying to send messages between nodes and is hitting cases in which sockets are not moving data in a timely manner, and the RTO on these connections is growing to a high value, the root cause of the problem that is leading to high RTOs must be debugged.

Possible issues that can lead to this symptom of connectivity problems and high TCP RTO values include the following examples:

- Packet loss in the network (not a IBM Spectrum Scale code issue)
- If flow control is enabled on the switches, flow control issues might be seen (for example, flow control taking effect when it should not be)
- A bug in the TCP stack (therefore, a base Linux issue)
- Some severe problem on either side of the connection that prevents ACKs from being turned around (all done in interrupt context); for example, memory is severely over-committed

Diagnosis and resolution

For more information about how to use information in IBM Spectrum Scale logs for debugging purposes, see "Debugging expels" on page 72.

The expels can be occurring because of packet loss on the network or issues with resource availability for networking on the node in the IBM Spectrum Scale cluster. The following sections provide more information about these issues that can cause expels or slow performance.

4.9 Understanding and resolving the effect of network packet loss

The network packet loss is a common cause of symptoms, such as node expels and performance problems. This section provides more information about how packet loss affects IBM Spectrum Scale environment and how to remediate it.

4.9.1 Overview and effect

In a distributed network of computing and storage nodes, interconnection between their applications is established through intermediate switches and routers. In a IBM Spectrum Scale cluster, various nodes open connections with each other to accomplish specific tasks. For example, when an I/O read/write operation is to be performed, many steps must occur before the I/O can be started between the IBM Spectrum Scale client and servers.

For example, to read a specific file, the token for the inode must be obtained from the node that keeps an in-memory copy of the inode (the metanode in IBM Spectrum Scale terminology). Also, to serialize access to data, IBM Spectrum Scale implements byte range locking; therefore, some handshaking might need to occur before access to a specific portion of a file is granted.

Packet loss is common in a typical network deployment; however, after packet loss exceeds acceptable limits, the effect on the entire cluster file system can become disastrous.

Consider the case of losing an import packet; for example, the request for a token that needed to perform a `stat()` operation on a file to be opened. The loss of important messages such as this can result in increasing latencies in accessing the cluster file system, and longer delays can stall the operation completely.

Therefore, if a node intends to open a file and read from it, dropping packets that are associated with gaining access to the inode stalls the ensuing read. However, in the case of a long sequential read (which can result in many pipelined packets coming back from the server), losing a single packet of read data does not stall the entire read operation.

Packet loss can occur for various reasons, such as network congestion, network buffer crunch in intermediate switches, or noise on a faulty or poorly plugged cable.

In an IBM Spectrum Scale cluster, packet loss can result in node expels and performance loss, as described in “Expels” on page 65. Even a short window of packet loss can have severe impacts.

For example, consider the case of two peers that have not yet communicated together, and do not have a socket established between them. If one peer node needs to request access to an inode that is managed by another node, a new socket must be opened to request the token. To establish a new socket, both sides must have the ARP entry for the other node, and transient packet loss in the network during the ARP resolution process can result in a connection failure, which can lead to an expel (by default, a new connection waits only three seconds for ARP resolution, depending on ARP tuning in effect, which should be defined by the **`sysctl mcast_solicit`** variable).

Therefore, to understand where a packet might be dropped between two communicating IBM Spectrum Scale nodes, it is helpful to understand how data is queued at various levels (for example, network adapter, operating system, and IBM Spectrum Scale code).

Figure 4-4 shows maps participating queues with troubleshooting tool or files for that layer.

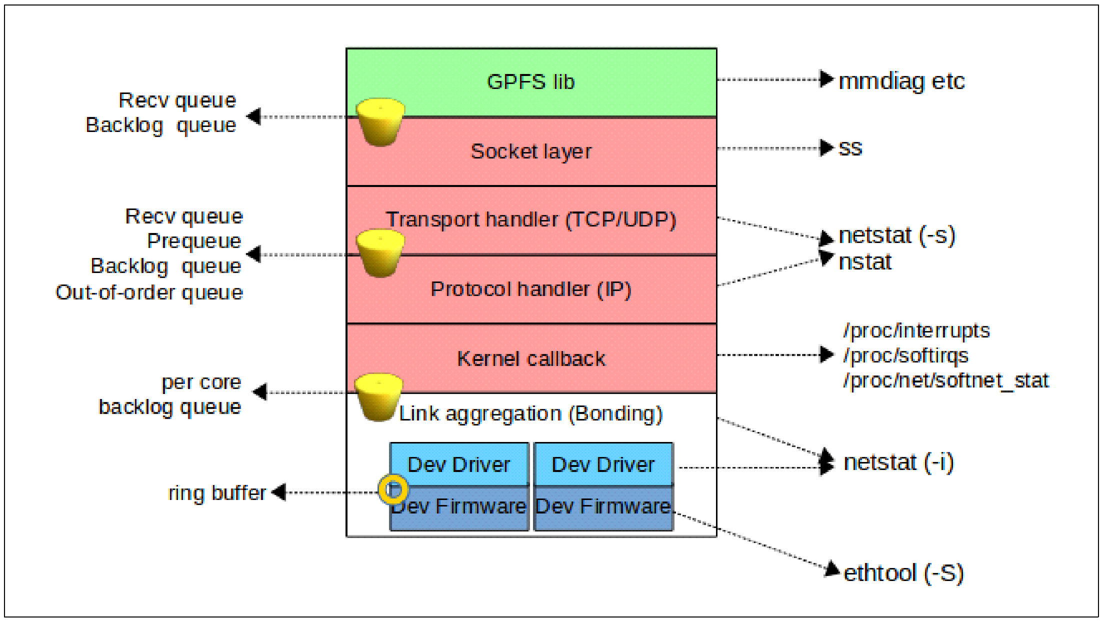


Figure 4-4 Participating queues with troubleshooting tool/files for each layer

As shown in Figure 4-4, multiple queues and buffers are involved, while packets manage to move between different layers. Therefore, we must choose a tool for inspecting the problem at the correct layer.

Manuals and various tutorials are available to explain various options for these tools. In the interest of the scope of this document, we explain how to use these tools to identify any packet loss and diagnose it, which otherwise can cause serious damage to your cluster arrangements.

Diagnosis and resolution

Referring to Figure 4-4 on page 75, inspecting interface statistics is the first place to check (or look) for the packet loss events. At the bottom most layer, device firmware records various statistics about received and transmitted packets in the private memory region of network device data structure, which is later accessed by **`ethtool`**, as an administrator executes it in the command line. **`Netstat`** and **`nstat`** displays various information about routing table, protocol layers, and interfaces that are configured in the server. These tools infer various stats about packets from Linux pseudo file system (/proc/net), which are updated by kernel as it moves the packets between network device (ring buffers) and protocol handling routines or vice versa.

As network device receives packets from wire and DMA them into its ring buffers, it is the job of the kernel to pull them into socket buffer (kernel datastruct to represent packets for higher network layers) as quickly as possible. Therefore, based on how kernel threads (per core softirq) are scheduled, considering the period for which these threads are allowed to pull the packets and overall load in the server, a chance exists when the kernel is slow to drain the packets from device's ring buffer and not keeping pace with packets rate at network link. This issue leads to an increased overrun count in **`netstat`**.

At the same time, the network device might also be experiencing packets burst (or attack) in the physical layer for various reasons (load imbalance from switch, flood attack, and so on), which can lead to an increased discard or drop of related count in **`netstat`** and **`ethtool`**. Because every vendor has their own implementation of counters on packet statistics, we try to generalize the causes that are common across most of the vendors.

Packet drop or discard is caused by the most common reasons:

- Insufficient ring buffers for DMA write
- Bottom-half kernel threads for RX/TX slower than top-half interrupts rate
- Insufficient budget for draining ring buffers

Insufficient ring buffers for DMA write

A typical network adapter reads the packets from its physical layer and writes them into its ring buffers so that the kernel driver for the adapter can start fetching them and let kernel network stack process them further. The design of ring buffers usage is that it can be written only by devices as and when it identifies a valid Ethernet frame.

Apart from packets that are participating in intended communication between servers, unsolicited packets can appear to the adapter, which eventually fills up slots in the ring buffer. This issue can d cause unnecessary interrupt service and packet handling by the kernel. In some extreme cases where they congest by using intended packets, they let applications (and hence its corresponding network layers) starve for data.

By default, most of the network drivers set up ring buffer counts to an optimal value and not to its possible values. Optimal values are used because too many acceptances at this circular buffer to slower kernel threads reading them cause buffer over-write. That adapter cannot hold the packets or even drop them when the kernel is still busy with something else, or not bothering to clear these buffers by promptly reading the packets.

Therefore, to avoid buffer over-run, they set an optimal value that is based on nominal interrupt rate for the adapter speed. You can use **`ethtool`** to increase this buffer size when you notice incrementing buffer over-run, discard, or drop-related counters (see Example 4-23 on page 77).

Example 4-23 ethtool usage examples

```
# Check packets drop through ethtool
 ethtool -S <interface> | egrep -i '(drop|discard|over)'

 # Check current ring buffers setting through ethtool
 ethtool -g <interface>
Ring parameters for enP2p1s0f0:
Pre-set maximums:
RX:   8192
RX Mini:0
RX Jumbo:0
TX:   8192
Current hardware settings:
RX:   1024
RX Mini:0
RX Jumbo:0
TX:   1024

 # Adjust the size of ring buffers using ethtool
ethtool -G <interface> rx 8192 tx 8192
```

Bottom-half kernel threads for RX/TX slower than top-half interrupts rate

In Linux, processing interrupt service for an interrupt is split into top-half and bottom-half, where in the former routine begins handling the hardware interrupt by performing the most critical and less time-consuming tasks for that interrupt.

Kernel softirq threads for Network RX/TX perform reading for packets from hardware ring buffers by making strictly a time-bound polling (NAPI) on these buffers at the same time, not beyond the user-configured budget. Most of the drivers implement reading up to 64 packets in one NAPI structure for a processor to read (also called *device weight*), but it is per CPU core backlog queue that decides quantum of received packets across all network devices that the CPU core scheduled for (also known as *netdev budget*).

If kernel softirq threads that perform NAPI polling on device cannot be scheduled at the right time or seem to be slower than the hardware interrupt rate (which depends on interrupt adaptive coalescence for that device), you notice an incrementing drop column in `/proc/net/softnet_stat`, which is based on the kernel version that is used. It might be helpful to increase (or double) the current setting to overcome this problem (see Example 4-24).

Example 4-24 Check for an incrementing drop counter

```
# Check for an incrementing drop counter. It is the second
# column for a kernel based on 3.1x. For other kernels, you may # check for
function softnet_seq_show() in its source.
cat /proc/net/softnet_stat

# Check current size of process backlog for softirqs.
sysctl net.core.netdev_max_backlog
1000

# Double the size of backlog processing.
sysctl -w net.core.netdev_max_backlog=2000
```

Insufficient budget for draining ring buffers

Contrary to the slower kernel threads scenario that was described in the previous section, a case might exist in which kernel softirq threads are at optimal scheduling, but are not given enough of a time to clear the ring buffers in the device. This issue occurs because an insufficient budget is configured for a specific network workload (see Example 4-25).

Example 4-25 Check for incrementing time squeeze counter and adjust size of backlog processing

```
# Check for an incrementing time squeeze counter. It is the
# third column for a kernel based on 3.1x. For other kernels,
# you can check for function softnet_seq_show() in its source.
cat /proc/net/softnet_stat

# Check current size of process backlog for softirqs.
sysctl net.core.netdev_budget
300

# If you are sure about more packets to read per NAPI call to
# the device, increment it from its default value.
sysctl net.core.dev_weight
64

# Double the size of backlog processing.
sysctl -w net.core.netdev_budget=600
```

After the packets are accepted by kernel for IP processing and then for TCP/UDP processing, a few places within TCP/UDP are available where the packets might be dropped because of back-pressure from an application (meaning that, application is reading its receive buffer at slower rate than the underlying network layer observing their rate of arrival), which can cause the receive queue in the underlying protocol stack to fill up to the maximum.

For TCP, it is more complicated because of the involvement of multiple queues within its implementation in kernel. Therefore, we must perform some careful analysis to inspect symptoms for packet drops.

Packet drop in TCP backlog queue

The TCP backlog queue serves to move packet data into the socket buffer when the application is busy reading and writing on the socket. When the socket receives a queue size with its backlog queue that is more than the socket's send and receive buffer size, you notice that the **`TCPBacklogDrop`** counter incrementing in **`nstat`**. When this problem occurs at a peak workload for your application, it is advisable to increment socket send and receive buffer size, as shown in the following example:

```
# Check for adequate buffer size for sockets
sysctl net.core.rmem_default
sysctl net.core.rmem_max
sysctl net.ipv4.tcp_rmem
```

Packet loss that results from firewall configuration

Packet loss can occur at various layers in the TCP/IP stack, depending on how the firewall is configured (for example, the IP layer or TCP layer). IBM Spectrum Scale needs access to specific TCP ports and ICMP protocol. For more information about TCP ports that are required by IBM Spectrum Scale, see the following IBM Knowledge Center web pages:

- IBM Spectrum Scale port usage
- IBM Spectrum Scale firewall configuration

Other reasons for slow response

After a packet is delivered by way of TCP or RDMA to the IBM Spectrum Scale daemon, the packet cannot be dropped; however, delays can occur within the IBM Spectrum Scale daemon in handling the request that is associated with the packet. You can use the **`mmdiag -waiters`** command to look for which requests are being waited on at the IBM Spectrum Scale layer. You also can query for statistics on the sockets that are used by IBM Spectrum Scale by running the **`mmdiag -network`** command.

For more information, see "Network troubleshooting with IBM Spectrum Scale mmdiag command" on page 50.

4.10 Resource issues

This section describes the need to balance the system resource consumption, such as CPU, memory, and network bandwidth, across multiple applications and IBM Spectrum Scale on all nodes in the cluster.

Overview and impact

As a cluster file system, IBM Spectrum Scale uses TCP/IP- or RDMA-based communication between various components, especially across NSD nodes. Restricting the scope of this documentation to TCP/IP in Linux network stack, every network layer in an operating system must operate at its best efficiency for IBM Spectrum Scale to perform at higher throughput for its storage applications. From components accessing sockets, the overall efficiency is measured on how sockets communicate each other across NSD and client nodes. Because IBM Spectrum Scale is installed, every network layer (including IBM Spectrum Scale configurations) optimally use system resources.

However, because other applications natively run along with IBM Spectrum Scale components in your operating system, sometimes it is required to inspect resource contention, being it CPU, memory, or even with network protocols, to diagnose and fix harming factors. For example, surprisingly a delay in renewing disk leases at cluster manager might seriously cause node expels if such delay is exceeding prescribed timeout for cluster manager to decide. Root cause for this delay might occur anywhere between these two nodes or within themselves before IBM Spectrum Scale components handling them.

Imbalance on interface usage

NIC bonding (or teaming) feature in the operating system to provide high availability and link aggregation over network interfaces to meet zero downtime for enterprise applications, such as IBM Spectrum Scale. Although multiple modes of bonding are available to configure, according to your network topology, a minor inspection is required over fairness of applied bonding mode to confirm whether it meets our requirement (for example, link aggregation).

Two common configurations are used for LACP: L2+L3 and L3+L4. L2+L3 applies hashing on Ethernet and IP header of packets to derive interface into which the packet is queued. Similarly, L3+L4 applies hashing on IP and TCP/UDP headers.

Because load imbalance on the subordinate interfaces is based header contents that is used for hashing, you must take corrective action by switching into another mode (or adjusting its hashing policy) if reduced throughput observed is for your IBM Spectrum Scale deployment.

If a lot of network traffic imbalance exists on the subordinate links, specific IBM Spectrum Scale packets that are more critical than regular packets can get delayed and cause issues.

This delay can adversely affect the performance or even the stability of the cluster or file system.

The consequence of delaying this data might even cause node expels and RPC-related timeouts, as described in “Expels” on page 65. Therefore, to check whether the hashing policy was chosen correctly, inspect the received packets ratio across the enslaved interfaces (see Example 4-26).

Example 4-26 Check for uniform distribution of packets handled in subordinates using netstat -i

```
# Check for uniform distribution of packets handled in subordinates
netstat -i

Kernel Interface table
Iface       MTU Met    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR
Flg
bond0      1500   0 11735259913      0      0      0        0      0      0      0
BMmRU
eth8       1500   0 6473065243      0      0      0        0      0      0      0
BMsRU
eth10      1500   0 5262194670      0      0      0        0      0      0      0
BMsRU
lo        65536   0  6433744      0      0      0  6433744      0      0      0
LRU
```

If too much bias exists on one subordinate that is handling most of the traffic, configure a different hashing policy.

Numa issues

Even after all of the optimizations are applied in the operating system and IBM Spectrum Scale deployments, a bad placement of the network adapter in the system board for which no memory bank exists locally severely slows down the IBM Spectrum Scale performance. Per-numa-node memory statistics must be carefully analyzed when you find that the adapter was misplaced by using information in the operation system, as shown in Example 4-27.

Example 4-27 Check the physical placement of the network adapter

```
# Check the physical placement of network adapter for its NUMA
# locality. If hwloc package is available, hwloc-ls could be
# used for quick reference. Else, refer to PCI ID of the
# adapter and check for its NUMA mapping by kernel in PCI.
hwloc-ls

Machine (128GB total)
  NUMANode L#0 (P#0 64GB)
    Package L#0
    <...>
    HostBridge L#0
      PCIBridge
        PCI 15b3:673c
          Net L#0 "ib0"
          Net L#1 "enp1s0d1"
          OpenFabrics L#2 "mlx4_0"
    HostBridge L#2
    <..>
  NUMANode L#1 (P#1 64GB)
```

```
    Package L#1
    <...>
    HostBridge L#6
      PCIBridge
        PCI 15b3:1013
          Net L#6 "enP2p1s0f0"
          OpenFabrics L#7 "mlx5_0"
        PCI 15b3:1013
          OpenFabrics L#8 "mlx5_1"
    HostBridge L#8
      PCIBridge
        PCIBridge
          PCIBridge
            PCI 14e4:168e
              Net L#9 "bootnet"
            PCI 14e4:168e
              Net L#10 "enP3p9s0f1"

# Refer in sysfs when you don't have hwloc package available.
cat /sys/class/net/<ethernet>/device/numa_node

# Check whether you have local memory bank attached. All numa
# nodes should have local memory.
numastat
```

The next step is to check whether your application can use memory from the corresponding numa node when it attempts to communicate with remote nodes through this specific network adapter. This guideline also applies to server-side applications, including IBM Spectrum Scale, as shown in the following example:

```
# Monitor for a right memory consumption from the numa nodes
watch -n1 -differences=cumulative -x numastat -c <application>
```

You can run a tool, such as **`nsdperf`**, as an application and check for its memory consumption. If you find most of the memory consumption is occurring not in local node for your application (that is, numa node where the application is running, through the available CPU cores), ensure that you set the CPU affinity for your applications in the correct set of CPUs for that numa node.

To ensure that IBM Spectrum Scale memory allocation is balanced across multiple NUMA nodes, set the IBM Spectrum Scale configuration parameter **`numaMemoryInterleave`** to value of **`yes`**. For more information, see IBM Knowledge Center.

Softirq balance on NIC interrupts

In “Bottom-half kernel threads for RX/TX slower than top-half interrupts rate” on page 77, we described how Linux handles interrupts from network devices. Kernel’s softirqd daemon tries to balance every CPU core for fair interrupt processing through per CPU `net.core.netdev_budget`.

However, in some cases, some CPU cores might be processing more packets than the remaining cores. This issue can occur if the CPU core that is scheduled to process the interrupt bottom-half is not same as the CPU core where application (socket) is listening for its data. Unless the kernel scheduler is aware of wise CPU cores to schedule (through kernel feature receive packet steering), fair scheduling by the kernel still can lead for imbalance transmit/receive packet processing by softirq:

```
# Check for uniform distribution of NET_TX/RX softirq
cat /proc/softirqs
```

If NET_TX and NET_RX are not balanced near equally across CPU cores, you might try disabling kernel softirqd service and manually pin network device interrupts with the list of CPUs if the application was pinned on them.

Another perspective of looking at this interrupt imbalance is to check whether multiple queues (each assigned per CPU) are running through any error that might amplify many other problems for packets that are handled on a specific queue. You can use **`ethtool -S`** for detailed multi-queue statistics, per the implementation of that device vendor.

The Mellanox adapter uses the following counters in CX4:

- queue_stopped and wake_queue must be equal in all ports. This equality is a good indication that the driver can drain packets from the kernel at all the times.
- vport_rx_unicast and vport_rx_dropped should be at a healthy ratio. Drops must be subminimal in comparison to vport_rx_unicast. This ratio confirms that rx packets are successfully delivered to higher protocol layers without any contention in software rx buffers (driver); therefore, the receive completion rate is in-phase with the hardware ingress rate.

A

IBM Spectrum Scale lease configuration variables effect on expel timing flows

This appendix describes the disk lease-related parameters that are used by **mmfsd** and how to display them. These lease-related configuration parameters influence how disk leases are handled by various nodes in the IBM Spectrum Scale cluster. You can display the current value of parameters by running the **mmfsadm** commands.

Note: Avoid running **mmfsadm dump** in production because a timing hole exposure can cause a daemon failure (even with **saferdump**). The **mmdiag** command provides much (but not all) of the **mmfsadm dump** data that might be needed in production.

The following example shows the use of the **mmfsadm saferdump** command:

```
# mmfsadm saferdump cfgmgr | grep -A 3 "^lease"
failureDetectionTime 35
no recoveryWait 35
dmsTimeout 23
leaseDuration 35.0/23.3
renewalInterval 30.0/11.7
renewalTimeout 5
fuzz 3.00/1.17
missedPingTimeout 15x2.0=30.0
totalPingTimeout 60x2.0=120.0
```

Lease configuration parameters related to disk leasing

This section describes the lease configuration parameters that are related to disk leasing.

failureDetectionTime

The default value that is used in the IBM Spectrum Scale daemon is 35 seconds (**`mmdiag`** reports the default as **`'-1'`**).

This variable controls how long it takes IBM Spectrum Scale to react to a node failure (how quickly the cluster manager might detect a failed node, expel it, and recover the cluster). When disk fencing (**`Persistent Reserve`**) is enabled, IBM Spectrum Scale can start faster recovery times. However, **`Persistent Reserve`** is not a typical configuration; therefore, we instead describe the more common “disk leasing” mechanism that is used to manage disk leases.

When disk leasing is enabled, **`failureDetectionTime`** can be changed by way of **`mmchconfig`** to set the length of the lease. Also, **`leaseDuration`** can be set directly by way of **`mmchconfig`** but large clusters typically include tuned **`failureDetectionTime`** instead, which allows **`leaseDuration`** to be derived from **`failureDetectionTime`**. Therefore, only this approach is considered here.

leaseDMSTimeout

The default value is 23 seconds (**`mmdiag`** reports the default as -1 and **`mmlsconfig`** reports the default as “2/3 **`leaseDuration`**”).

When a IBM Spectrum Scale server node acquires a disk lease, it schedules a timeout interrupt to occur **`leaseDMSTimeout`** seconds after the new lease is set to expire. When that timer pops, if the lease associated with the timer was not renewed, IBM Spectrum Scale starts a kernel panic if any local I/O is pending to a cluster file system for which the server’s lease expired.

leaseDuration

The default value for the duration of a IBM Spectrum Scale disk lease is 35 seconds (**`mmdiag`** reports the default as **`-1`**).

It is highly recommend that this variable never be directly set or tuned (use **`failureDetectionTime`** instead to tune this variable).

The **`mmfsadm saferdump`** command (which should be avoided on production systems) displays two values for this configuration option: first, the length of non-quorum node leases, and second, the length of quorum node leases.

Quorum nodes must renew their disk leases more frequently (two thirds of the non-quorum node lease values) because the quorum nodes’ renewal mechanism also verifies the existence of a valid cluster manager. This shorter lease facilitates the quorum nodes’ election of a replacement cluster manager when required.

leaseRecoveryWait

The default value is 35 seconds. To maintain the integrity of the file system, tuning this value to less than 35 seconds is discouraged unless it can be ensured that I/Os are never delayed.

The **`leaseRecoveryWait`** configuration option is used in two ways by the IBM Spectrum Scale code when disk leasing is enabled (as opposed to Persistent Reserve, which is not covered here):

- The default value of **`leaseDMSTimeout`** is calculated to be two thirds of **`leaseRecoveryWait`**.
- The value of **`leaseRecoveryWait`** is used by the IBM Spectrum Scale daemon to determine how long to wait before running recovery protocols (also called log recovery) after an expel event occurs. Because recovery must always wait at least **`leaseRecoveryWait`** seconds after a node is expelled, IBM Spectrum Scale ensures that **`missedPingTimeout`** can never be smaller than **`leaseRecoveryWait`**.

renewalInterval

The default value is 30 seconds; **`renewalInterval`** is derived by subtracting **`renewalTimeout`** (which is by default 5 seconds) from **`leaseDuration`** (which defaults to 35 seconds).

The **`renewalInterval`** tunable defines the maximum number of seconds a node waits after it obtains a valid lease before it attempts to renew that lease. On each new lease renewal, a random *fuzz factor* (see “renewalTimeout”) is subtracted from this **`renewalInterval`** value, in an attempt to stagger lease renewal times across the nodes.

renewalTimeout

The default value is 5 seconds, and always is set to 5, unless the value of **`leaseDuration`** is set to less than 10 seconds. If **`leaseDuration`** is set to less than 10 seconds, **`renewalTimeout`** is set to one half of **`leaseDuration`**.

This value is used in calculating the interval in which the lease is renewed.

The **`fuzz`** factor is the default value set to 3 seconds (derived from one-tenth of the **`renewalInterval`**, which defaults to 30).

Per the **`renewalInterval`** description, after each new lease renewal, the number of seconds a node waits before starting a new lease renewal is calculated to be **`renewalInterval`** seconds, minus a random *fuzz factor*, which is defined to be a random number of seconds between zero and this **`fuzz`** configuration option.

Therefore, with the default tuning, a node attempts to renew its lease on an interval between (**`renewalInterval - fuzz`** = 30-3) 27 and (**`renewalInterval`**) 30 seconds.

maxMissedPingTimeout

The default value is 60 seconds.

When a node fails to renew its lease within the allotted **`leaseDuration`** window, the cluster manager pings (send ICMP datagrams to) the node (this process is sometimes referred to the *ping timeout window*). One of the goals of these pings is to determine if a node still has a good connection to the cluster manager, and therefore might be allotted more time to renew its lease before an expel.

The **maxMissedPingTimeout** is used to define a maximum value for IBM Spectrum Scale daemon's **missedPingTimeout** variable, which defines the length of time the cluster manager pings before expelling a node whether the pings fail. The value of the internal **missedPingTimeout** variable is calculated as described next.

minMissedPingTimeout

The default value of the tunable is 3 seconds.

The value of **minMissedPingTimeout** defines a minimum value for IBM Spectrum Scale daemon's **missedPingTimeout** variable. The actual value of the internal **missedPingTimeout** variable is taken to be the maximum of **minMissedPingTimeout** and **leaseRecoveryWait** (but it is limited to a maximum value that is defined by **maxMissedPingTimeout**).

IBM Spectrum Scale uses the daemon **missedPingTimeout** value to define the amount of time the cluster manager waits before expelling a node with an overdue lease if the node does not respond to the cluster manager's pings.

Note: With the default tuning, **missedPingTimeout** is defined by **leaseRecoveryWait**.

totalPingTimeout

The default value is 120 seconds.

This tunable is used by the IBM Spectrum Scale daemon to define the total amount of time the cluster manager waits before expelling a node with an overdue lease if the node responds to the pings that are sent by the cluster manager. Because this tunable is set to 120 seconds by default, it is generally left at the default.

The intent of having separate values for **totalPingTimeout** and **missedPingTimeout** (which are defined according to the values of **minMissedPingTimeout**, **maxMissedPingTimeout**, and **leaseRecoveryWait**) allows for tuning such that nodes that still have good network connectivity back to the cluster manager are allotted more tolerance in terms of slow lease responses back to the cluster manager.

B

nsdperf command examples

This appendix provides **nsdperf** examples to assess the network bandwidth for multiple client/server scenarios across TCP/IP and RDMA protocol. In the example that is presented, the client and server are interconnected by using FDR InfiniBand (FDR-IB), with one 1 x FDR-IB link per node. The `ib0` suffix to the node name denotes the IP address that corresponds to the IP over InfiniBand interface (IPoIB).

Comments inline in the **nsdperf** examples are denoted by “#” at the start of the line. This appendix assumes that the `nsdperf` (`nsdperf-ib`) binary is installed on all the nodes in `/opt/benchmarks/` directory.

This appendix includes the following topics:

Single Client and Single Server

In Example B-1, the network bandwidth between node `c71f1c7p1ib0` and `c71f1c9p1ib0` over the TCP/IP and RDMA network is assessed. The “nsdperf in server mode” is started in the client and server node:

```
mmdsh -N c71f1c7p1ib0,c71f1c9p1ib0 "/opt/benchmarks/nsdperf-ib -s
</dev/null>/dev/null 2>&1 &"
```

Then, run **`nsdperf`** (see Example B-1) from an administrative node (for example, log-in node, gateway node, or any cluster node that permits interactive job execution.

Example B-1 nsdperf example with single client and single server

```
# /opt/benchmarks/nsdperf-ib
# Designate the nodes as clients using "client" parameter
nsdperf-ib> client c71f1c7p1ib0
Connected to c71f1c7p1ib0
# Designate the nodes as servers using "server" parameter
nsdperf-ib> server c71f1c9p1ib0
Connected to c71f1c9p1ib0
# Set the run time to 30 seconds for the tests using "ttime" parameter
nsdperf-ib> ttime 30
Test time set to 30 seconds
# Perform the desired nsdperf network tests using "test" parameter.
# TCP/IP network mode - Use "status" command to verify client node connectivity to the server #
node
nsdperf-ib> status
test time: 30 sec
data buffer size: 4194304
TCP socket send/receive buffer size: 0
tester threads: 4
parallel connections: 1
RDMA enabled: no
clients:
c71f1c7p1ib0 (10.168.117.199) -> c71f1c9p1ib0
servers:
c71f1c9p1ib0 (10.168.117.205)
# Perform performance tests from clients to servers.
# The "test" command sends a message to all clients nodes to begin network performance testing
# to the server nodes. By default, write and read tests are performed.

nsdperf-ib> test
1-1 write 3170 MB/sec (756 msg/sec), cli 2% srv 3%, time 30, buff 4194304
1-1 read 3060 MB/sec (728 msg/sec), cli 3% srv 2%, time 30, buff 4194304
```

In Example B-1, the aggregate TCP/IP write bandwidth between 1 `nsdperf` client and 1 `nsdperf` server1 (each node with 1 x FDR-IB link) is 3170 MBps for 4194034 bytes (4 MiB) buffer size. The message rate for 4 MiB buffer size is 756 ps.

The **`nsdperf`** test duration is 30 seconds. The CPU busy (non-idle) percentage is 3%, on average for all the `nsdperf-server` nodes and 2%, on average for all the client nodes.

The aggregate TCP/IP read bandwidth between one `nsdperf` client and one `nsdperf server1` (each node with 1 x FDR-IB link) is 3060 MBps for 4194034 bytes (4 MiB) buffer size.

The message rate for 4 MiB buffer size is 728 ps. The **nsdperf** test duration is 30 seconds. The CPU busy (non-idle) percentage is 2%, on average for all the server nodes and 3%, on average for all the client nodes.

Example B-2 shows `nsdperf` execution between one server (such as NSD server) and one client (such as IBM Spectrum Scale Posix client) with RDMA that is used to send the data blocks.

Example B-2 nsdperf network tests between one client and one server

```
# Enable RDMA for sending data blocks using "rdma" parameter
nsdperf-ib> rdma on
RDMA is now on
# Perform the desired nsdperf network tests using "test". Default is write and read tests
# RDMA network mode - Use "status" command to verify client node RDMA connectivity to
# the server node
nsdperf-ib> status
test time: 30 sec
data buffer size: 4194304
TCP socket send/receive buffer size: 0
tester threads: 4
parallel connections: 1
RDMA enabled: yes
clients:
c71f1c7p1ib0 (10.168.117.199) -> c71f1c9p1ib0
mlx5_0:1 10a2:1f00:032d:1de4
servers:
c71f1c9p1ib0 (10.168.117.205)
mlx5_0:1 40a1:1f00:032d:1de4
# Perform RDMA performance tests using "test". Default is write and read tests
nsdperf-ib> test
1-1 write 6450 MB/sec (1540 msg/sec), cli 1% srv 1%, time 30, buff 4194304, RDMA
1-1 read 6450 MB/sec (1540 msg/sec), cli 1% srv 1%, time 30, buff 4194304, RDMA
# Based on results, RDMA bandwidth is limited by 1 x FDR-IB link (1-1) between the client and
# server [refer to APPENDIX B].
# By default, each client node uses four tester threads. Each of these threads will
independently
# send and receive messages to the server node. The thread counts can be scaled using "threads"
# parameter
# Shut down nsdperf (in server mode) on all client and server nodes using "killall" command
nsdperf-ib> killall
# Exit from program using "quit" command
nsdperf-ib> quit
```

Multiple clients and servers

In Example B-3, the network bandwidth between multiple client nodes `c71f1c9p1ib0` and `c71f1c10p1ib0`, and multiple server nodes `c71f1c7p1ib0` and `c71f1c8p1ib0` over the TCP/IP and RDMA network is assessed. The “nsdperf in server mode” is started in the client and server nodes:

```
mmdsh -N c71f1c7p1ib0,c71f1c8p1ib0,c71f1c9p1ib0,c71f1c10p1ib0
"/opt/benchmarks/nsdperf-ib -s </dev/null >/dev/null 2>&1 &"
```

Then, run **`nsdperf`** (see Example B-3) from an administrative node (for example, log-in node, gateway node, or any cluster node that permits interactive job execution).

In IBM Spectrum Scale environments, multiple IBM Spectrum Scale Posix clients send and receive network traffic from multiple NSD servers. Therefore, you must test `nsdperf` between multiple clients and servers. Example B-3 shows how to perform `nsdperf` tests between multiple clients and servers simultaneously.

Example B-3 nsdperf test between multiple client nodes and multiple server nodes

```
# /opt/ benchmarks/nsdperf-ib
# Designate the nodes as servers using the "server" parameter
nsdperf-ib> server c71f1c7p1ib0 c71f1c8p1ib0
Connected to c71f1c7p1ib0
Connected to c71f1c8p1ib0
# Designate the nodes as clients using the "client" parameter
nsdperf-ib> client c71f1c9p1ib0 c71f1c10p1ib0
Connected to c71f1c9p1ib0
Connected to c71f1c10p1ib0
# Set run time to 30 seconds for the tests using the "ttime" parameter
nsdperf-ib> ttime 30
Test time set to 30 seconds
# Perform the desired nsdperf network tests using the "test" parameter.
# TCP/IP network mode - Use "status" command to verify client node connectivity to the server
node
nsdperf-ib> status
test time: 30 sec
data buffer size: 4194304
TCP socket send/receive buffer size: 0
tester threads: 4
parallel connections: 1
RDMA enabled: no
clients:
c71f1c9p1ib0 (10.168.117.205) -> c71f1c7p1ib0 c71f1c8p1ib0
c71f1c10p1ib0 (10.168.117.208) -> c71f1c7p1ib0 c71f1c8p1ib0
servers:
c71f1c7p1ib0 (10.168.117.199)
c71f1c8p1ib0 (10.168.117.202)
# Perform performance tests from clients to servers.
nsdperf-ib> test
2-2 write 8720 MB/sec (2080 msg/sec), cli 3% srv 5%, time 30, buff 4194304
2-2 read 10200 MB/sec (2440 msg/sec), cli 5% srv 3%, time 30, buff 4194304
# Based on the results, TCP/IP bandwidth is limited by 2 x IPoIB link (2-2) between the clients
# and servers
# Enable RDMA for sending data blocks using "rdma" parameter
nsdperf-ib> rdma on
```

```
RDMA is now on
# Perform the desired nsdperf network tests using "test". Default is write and read tests
# RDMA network mode - Use "status" command to verify client node RDMA connectivity to
# the server node
nsdperf-ib> status
test time: 30 sec
data buffer size: 4194304
TCP socket send/receive buffer size: 0
tester threads: 4
parallel connections: 1
RDMA enabled: yes
clients:
c71f1c9p1ib0 (10.168.117.205) -> c71f1c7p1ib0 c71f1c8p1ib0
mlx5_0:1 40a1:1f00:032d:1de4
c71f1c10p1ib0 (10.168.117.208) -> c71f1c7p1ib0 c71f1c8p1ib0
mlx5_0:1 e0a5:1f00:032d:1de4
servers:
c71f1c7p1ib0 (10.168.117.199)
mlx5_0:1 10a2:1f00:032d:1de4
c71f1c8p1ib0 (10.168.117.202)
mlx5_0:1 e0a1:1f00:032d:1de4
# Perform RDMA performance tests using "test". Default is write and read tests
nsdperf-ib> test
2-2 write 12900 MB/sec (3080 msg/sec), cli 1% srv 1%, time 30, buff 4194304, RDMA
2-2 read 12900 MB/sec (3080 msg/sec), cli 1% srv 1%, time 30, buff 4194304, RDMA
# Based on the results, RDMA bandwidth is limited by 2 x FDR-IB link (2-2) between the clients
# and servers
# Shut down nsdperf (in server mode) on all client and server nodes using "killall" command
nsdperf-ib> killall
# Exit from program using "quit" command
nsdperf-ib> quit
```

Supplemental nsdperf tests and commands

This section describes supplement **nsdperf** tests (for example, **nwrite**) and commands (for example, **buffsize**, **hist**).

In Figure B-4, the network bandwidth between multiple client nodes `c71f1c9p1ib0` and `c71f1c10p1ib0` and multiple server nodes `c71f1c7p1ib0` and `c71f1c8p1ib0` over the RDMA network is assessed.

The “nsdperf in server mode” is started in the client and server nodes:

```
mmdsh -N c71f1c7p1ib0,c71f1c8p1ib0,c71f1c9p1ib0,c71f1c10p1ib0
"/opt/benchmarks/nsdperf-ib -s </dev/null >/dev/null 2>&1 &"
```

Then, run **nsdperf** (see Example B-4_ from an administrative node (for example, log-in node, gateway node, or any cluster node that permits interactive job execution).

Example B-4 shows the scenario with multiple clients and multiple server running **nsdperf** and ways to control time, RDMA, and viewing status or values of different parameters that are provided to **nsdperf**.

Example B-4 nsdfperf tests using the test parameter

```
# /opt/ benchmarks/nsdperf-ib
nsdperf-ib> server c71f1c7p1ib0 c71f1c8p1ib0
Connected to c71f1c7p1ib0
Connected to c71f1c8p1ib0
nsdperf-ib> client c71f1c9p1ib0 c71f1c10p1ib0
Connected to c71f1c9p1ib0
Connected to c71f1c10p1ib0
# Set the run time to 30 seconds for the tests
nsdperf-ib> ttime 30
Test time set to 30 seconds
# Enable RDMA for sending data blocks
nsdperf-ib> rdma on
RDMA is now on
# Perform the desired nsdperf network tests using the "test" parameter.
# RDMA network mode - Use "status" command to verify client node RDMA connectivity to
# the server node
nsdperf-ib> status
test time: 30 sec
data buffer size: 4194304
TCP socket send/receive buffer size: 0
tester threads: 4
parallel connections: 1
RDMA enabled: yes
clients:
c71f1c9p1ib0 (10.168.117.205) -> c71f1c7p1ib0 c71f1c8p1ib0
mlx5_0:1 40a1:1f00:032d:1de4
c71f1c10p1ib0 (10.168.117.208) -> c71f1c7p1ib0 c71f1c8p1ib0
mlx5_0:1 e0a5:1f00:032d:1de4
servers:
c71f1c7p1ib0 (10.168.117.199)
mlx5_0:1 10a2:1f00:032d:1de4
c71f1c8p1ib0 (10.168.117.202)
mlx5_0:1 e0a1:1f00:032d:1de4
# Perform the desired nsdperf network tests using the "test" parameter.
nsdperf-ib> test
2-2 write 12900 MB/sec (3080 msg/sec), cli 1% srv 1%, time 30, buff 4194304, RDMA
2-2 read 12900 MB/sec (3080 msg/sec), cli 1% srv 1%, time 30, buff 4194304, RDMA
# Perform individual network tests (for example, nwrite)
nsdperf-ib> test write
2-2 write 12900 MB/sec (3080 msg/sec), cli 1% srv 1%, time 30, buff 4194304, RDMA
nsdperf-ib> test read
2-2 read 12900 MB/sec (3080 msg/sec), cli 1% srv 1%, time 30, buff 4194304, RDMA
nsdperf-ib> test nwrite
2-2 nwrite 12900 MB/sec (3080 msg/sec), cli 1% srv 1%, time 30, buff 4194304, RDMA
```

Based on the results, RDMA bandwidth is limited by 2 x FDR-IB link (2-2) between the clients and servers.

Example B-5 shows the scenario in which the “hist parameter” is turned on (`hist on`), and `nsdperf` reports the bandwidth and the network response histogram.

Example B-5 nsdperf with hist parameter turned on

```
# The hist parameter can be turned "on" to print the network response time histograms
nsdperf-ib> hist on
Histogram printing is now on
nsdperf-ib> test write
2-2 write 12900 MB/sec (3080 msg/sec), cli 1% srv 1%, time 30, buff 4194304, RDMA
c71f1c9p1ib0 block transmit times (average 2.598 msec, median 3 msec)
msec nevents
1 2
2 1211
3 35724
4 2
c71f1c10p1ib0 block transmit times (average 2.598 msec, median 3 msec)
msec nevents
2 263
3 36674
4 1
```

In Example B-5, the aggregate RDMA write bandwidth between two `nsdperf` clients and two `nsdperf` servers (each node with 1 x FDR-IB link) is 12900 MBps for 4194034 bytes (4 MiB) buffer size. The RDMA message rate for 4MiB buffer size is 3080 ps.

The `nsdperf` test duration is 30 seconds. The CPU busy (non-idle) percentage is 1%, on average for all the `nsdperf -server` nodes and 1% on average for all the nsdperf-client nodes.

The `nsdperf` RDMA aggregate bandwidth is limited by 2 x FDR InfiniBand links (2 - 2) between the clients and servers. Based on the response time histogram, each of the clients have similar average and median response time. Comparing the histogram outputs for various clients can be useful to determine the slow performing clients.

In Example B-5, the nodes `c71f1c9p1ib0` and `c71f1c10p1ib0` have most of the block transmit times 2 - 3 ms range and therefore are performing equally.

In IBM Spectrum Scale environments, the control messages are small and must complete with low latencies. Example B-6 on page 94 shows `nsdperf` tests with multiple clients and histogram turned on for small message send. Based on response time histogram, each of the clients shows similar average and median response times (see Example B-6).

Example B-6 nsdperf tests with multiple clients and histogram turned on for small message send

```
# Set the buffsize to 1 byte to assess the network latency for small messages
nsdperf-ib> buffsize 1
Buffer size set to 1 bytes
nsdperf-ib> test write
2-2 write 1.27 MB/sec (74800 msg/sec), cli 4% srv 4%, time 30, buff 1, RDMA
c71f1c9p1ib0 block transmit times (average 0.1124 msec, median 0 msec)
msec nevents
0 850036
1 21
2 4
3 5
c71f1c10p1ib0 block transmit times (average 0.1012 msec, median 0 msec)
msec nevents
0 944850
1 9
# Based on the response time histogram, each of the clients have similar average and median
# response time. This can be useful to isolate any slow performing clients.
# Shut down nsdperf (in server mode) on all client and server nodes
nsdperf-ib> killall
# Exit from program
nsdperf-ib> quit
```

Related publications

The publications that are listed in this section are considered particularly suitable for a more detailed discussion of the topics that are covered in this paper.

IBM Redbooks

The following IBM Redbooks publications provide more information about the topic in this document. Note that some publications that are referenced in this list might be available in soft copy only:

- *IBM Elastic Storage Server Implementation Guide for Version 5.3,* REDP-5487
- *IBM Spectrum Scale Erasure Code Edition: Planning and Implementation Guide*, REDP-5557
- *Monitoring and Managing IBM Spectrum Scale Using the GUI,* REDP-5458
- *Monitoring and Managing the IBM Elastic Storage Server Using the GUI*, REDP-5471
- *Implementing IBM Spectrum Scale*, REDP-5254
- *Introduction Guide to the IBM Elastic Storage System,* REDP-5253
- *Implementation Guide for IBM Elastic Storage System 3000*, SG24-8443

You can search for, view, download, or order these documents and other Redbooks, Redpapers, Web Docs, draft, and additional materials, at the following website:

ibm.com/redbooks

Online resources

The following websites are also relevant as further information sources:

- IBM Knowledge Center
- The recently developed IBM Spectrum Scale Network readiness tool for IBM Spectrum Scale ECE can also be used for all IBM Spectrum Scale installations. You can find more details about the tool at:

  https://github.com/IBM/SpectrumScale_NETWORK_READINESS#start-of-content

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

REDP-5484-00

ISBN 0738459410

Printed in U.S.A.