LBNL-52456

The Global Unified Parallel File System (GUPFS) Project: FY 2002 Activities and Results

Gregory F. Butler,[1] Rei Chi Lee,[1] and Michael L. Welcome[2]

[1]NERSC Center Division
[2]Computational Research Division
Ernest Orlando Lawrence Berkeley National Laboratory
Berkeley, CA 94720

April 7, 2003

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computer Research, Mathematical, Information, and Computational Sciences Division, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

Table of Contents

The Global Unified Parallel File System (GUPFS) Project: FY 2002 Activities and Results

Executive Summary

The Global Unified Parallel File System (GUPFS) project is a multiple-phase, five-year project at the National Energy Research Scientific Computing (NERSC) Center to provide a scalable, high performance, high bandwidth, shared file system for all the NERSC production computing and support systems. The primary purpose of the GUPFS project is to make it easier to conduct advanced scientific research using the NERSC systems. This is to be accomplished through the use of a shared file system providing a unified file namespace, operating on consolidated shared storage that is directly accessed by all the NERSC production computing and support systems.

In order to successfully deploy a scalable high-performance shared file system with consolidated disk storage, three major emerging technologies must be brought together: (1) shared/cluster file systems, (2) cost-effective, high performance storage area network (SAN) fabrics, and (3) high performance storage devices. Although they are evolving rapidly, these emerging technologies individually are not targeted towards the needs of scientific high performance computing (HPC). The GUPFS project is intended to evaluate these emerging technologies to determine the best combination of solutions for a center-wide shared file system, to encourage the development of these technologies in directions needed for HPC at NERSC, and to then put them into production.

During the first three years of the GUPFS project, NERSC intends to test, evaluate, and influence the development of the technologies necessary for the successful deployment of a center-wide shared file system. Provided that an assessment of the technologies is favorable at the end of the first three years, the last two years of the GUPFS project will focus on a staged deployment, leading to production in FY 2006.

During its first year, FY 2002, the GUPFS project focused on identifying, testing, and evaluating existing and emerging shared/cluster file system, SAN fabric, and storage technologies; identifying NERSC user input/output (I/O) requirements, methods, and mechanisms; and developing appropriate benchmarking methodologies and benchmark codes for a parallel environment. This report presents the activities and progress of the GUPFS project during its first year, the results of the evaluations conducted, and plans for near-term and longer-term investigations.

Shared file system technology has been undergoing turbulent changes during the past year. Some once-promising shared file systems seem to be fading; other existing shared file systems are continuing to make incremental progress; many new shared file system vendors and alternative technology approaches are appearing; and a major shared file system development project was awarded. Some highlights of these changes include:

- The Sistina GFS file system continues to make limited advances, but the commercial market segment to which it is targeted is becoming more and more remote from scientific HPC.

- ADIC is continuing to develop their CentraVision File System (CVFS), which will soon to be renamed StorNext. ADIC is interested in working with NERSC and other DOE laboratories to make CVFS work in the HPC environment and to integrate it with HPSS.

- IBM's GPFS file system continues to advance, both on large SP systems and on IBM Linux clusters.

- Many new shared file system vendors are appearing, some with interesting and promising technologies. These include Maximum Throughput with their local area network (LAN)-distributed InfinARRAY file system, Ibrix with their somewhat similar file system, and a host of other vendors including 3PAR, PolyServe, and SANique.

- The Advanced Simulation and Computing (ASCI) PathForward program awarded the Scalable Global Secure File System (SGSFS) development contract to Hewlett-Packard for the development of the Lustre file system. Since SGSFS shares a number of the same endpoint functionality goals as GUPFS, Lustre is likely to be one of the candidate shared file system solutions for GUPFS. However, NERSC also has requirements that are not fully reflected in the ASCI workload, so it remains to be seen if Lustre can satisfy all the requirements of each workload.

A new category of storage systems called *utility storage* is currently being introduced by many storage startups such as 3PRAdata and YottaYotta. Through innovative hardware and software design, these storage startups have promised to deliver a higher level of scalability, connectivity, and performance. The building blocks of utility storage can scale from a few dozen to a few hundred terabytes, or even in principle to the petabyte range, with a transfer rate as fast as a few thousand megabytes per second and the ability to sustain beyond 100,000 I/O operations per second (IOPS) for online transaction processing (OLTP) applications. While storage startups are introducing new storage technologies, the existing storage vendors are also introducing storage systems supporting 2 Gb/s Fibre Channel ports and an increased number of ports for SAN connectivity.

One of the major storage issues encountered during the FY 2002 evaluation is scalability—how well the storage is able to process simultaneous I/O requests from multiple clients to a single device. Our evaluation found that scalability of single-device access is a common problem in many storage systems, in terms of both the number of clients allowed on a single device and the performance of shared access. Although we believe that the storage vendors will address these issues before GUPFS is ready for deployment, we will continue monitoring developments in this area and work with the vendors to steer their development effort to support the needed functionalities.

Fibre Channel technology has recently been upgraded from 1 Gb/s to 2 Gb/s bandwidth, which will allow substantially improved storage performance to be realized. Other SAN Fabric technologies are also beginning to appear:

- Using Ethernet as a SAN fabric is now becoming possible due to the iSCSI standard, which is a block storage transport protocol. The iSCSI protocol allows the standard SCSI packets

to be enveloped in IP packets and transported over standard Ethernet infrastructure, which allows SANs to be deployed on IP networks. This option is very attractive as it allows lower-cost SAN connectivity than can be achieved with Fibre Channel, although with lower performance. It will allow large numbers of inexpensive systems to be connected to the SAN and use the cluster file system through commodity-priced components. While attractive from a hardware cost perspective, this option does incur a performance impact on each host due to increased traffic through the host's IP stack.

- The newly emerging InfiniBand interconnect also shows promise for transporting storage traffic in a SAN. InfiniBand offers performance (bandwidth and latency) beyond that of either Ethernet or Fibre Channel, with even higher bandwidths planned.

Building on the achievements of FY 2002, the GUPFS project in FY 2003 will focus on investigating and evaluating additional shared file system technologies with an emphasis on aggregate performance and file system scalability. The GUPFS project will also focus on demonstrating 1 GB/s sustained file system performance, and on evaluating the performance characteristics of SAN fabric technologies and their interoperability in a hybrid, multiple-fabric environment. In FY 2003, the GUPFS project plans to:

- evaluate the ADIC StorNext (CVFS) shared file system
- evaluate Maximum Throughput's InfinARRAY shared file system
- evaluate the Lustre object-oriented shared file system
- attempt to demonstrate 1 GB/s aggregate sustained I/O transfers with shared file systems
- evaluate the iSCSI protocol for accessing storage over IP networks
- evaluate the Infiniband interconnect as a SAN fabric
- evaluate 2 GB/s Fibre Channel
- evaluate the combination of Gigabit Ethernet, Infiniband, and Fibre Channel into a single hybrid fabric.

1 Introduction

The Global Unified Parallel File System (GUPFS) project is a multiple-phase, five-year project at the National Energy Research Scientific Computing (NERSC) Center to provide a scalable, high performance, high bandwidth, shared file system for all the NERSC production computing and support systems. The primary purpose of the GUPFS project is to make it easier to conduct advanced scientific research using the NERSC systems. This is to be accomplished through the use of a shared file system providing a unified file namespace, operating on consolidated shared storage that is directly accessed by all the NERSC production computing and support systems.

This report relates the activities and progress of the GUPFS project during its first year, FY 2002. It also presents the results of the evaluations conducted during FY 2002, as well as plans for near-term and longer-term investigations.

1.1 GUPFS Project Overview

The primary goal of the GUPFS project is to increase the capability of the NERSC client community to conduct scientific research by simplifying and unifying the storage resources at NERSC. The GUPFS project intends to accomplish this by introducing a scalable, high-performance, high-bandwidth shared file system into the NERSC production environment. This file system will replace all the individual system-specific user data file systems with common shared file systems residing on consolidated disk storage. The benefits of such a file system will include a unified file namespace, no need to transfer files between systems, and visibility and accessibility of all files from all of the NERSC production and support systems. Consolidating the storage combines the benefit of central storage management with the ability to incrementally increase storage capacity and make all of the storage available to every system.

In order to successfully deploy a scalable high-performance shared file system with consolidated disk storage, three major emerging technologies must be brought together: (1) shared/cluster file systems, (2) cost-effective, high performance storage area network (SAN) fabrics,[1] and (3) high performance storage devices. Although they are evolving rapidly, these emerging technologies individually are not targeted towards the needs of scientific high performance computing (HPC). The GUPFS project is intended to evaluate these emerging technologies to determine the best combination of solutions for a center-wide shared file system, and to encourage the development of these technologies in directions needed for HPC at NERSC.

The GUPFS project started in the last half of FY 2001 as a limited investigation of the suitability of shared-disk file systems in a SAN environment for scientific clusters, with an eye towards possible future center-wide deployment. As such, it was targeted towards initial testing of the Sistina Global File System (GFS), and included a small testbed system to be used in the investigation.

[1] In this context, the term *fabric* refers generally to a complex networked topology; more specifically, it means one or more switches in a network topology, connecting one or more hosts (initiators) with one or more storage devices (targets).

With the advent of the NERSC Strategic Proposal for FY 2002–2006 [1], this modest investigation evolved into the GUPFS project, which is one of the major programmatic thrusts at NERSC. During the first three years of the GUPFS project, NERSC intends to test, evaluate, and influence the development of the technologies necessary for the successful deployment of a center-wide shared file system. Provided that an assessment of the technologies is favorable at the end of the first three years, the last two years of the GUPFS project will focus on a staged deployment of a high performance shared file system center-wide at NERSC, in conjunction with the consolidation of user disk storage, leading to production in FY 2006.

1.2 FY 2002 Activity Overview

During its first year, the GUPFS project focused on identifying, testing, and evaluating existing and emerging shared/cluster file system, SAN fabric, and storage technologies; identifying NERSC user input/output (I/O) requirements, methods, and mechanisms; and developing appropriate benchmarking methodologies and benchmark codes for a parallel environment.

With the change in scope from the initial shared-disk file system evaluation to the more comprehensive GUPFS project, a substantial portion of the first year was consumed by activities relating to planning, organizing, and starting up the new project. However, the GUPFS project was able to conduct a significant number of other activities during FY 2002:

- Completed the acquisition and installation of the initial GFS testbed at the beginning of FY 2002.
- Configured the GFS testbed to test file systems in a scientific parallel computational cluster environment.
- Realigned the original GFS testbed project and developed the GUPFS project as part of the NERSC Strategic Proposal.
- Determined an appropriate project plan for GUPFS and assigned/acquired resources for the project.
- Presented the GUPFS project purpose, goals, and plan to DOE and other organizations.
- Developed relationships with file system, SAN fabric, and storage vendors, as well as other laboratories and possible collaborators.
- Identified and tracked existing and emerging file system, SAN fabric, and storage technologies and technology trends.
- Developed testing methodologies and benchmarks for shared/cluster file systems in a parallel environment.
- Obtained baseline storage performance characteristics and conducted initial testing of two versions of the Sistina GFS file system.
- Expanded the testbed to extend scalability testing capabilities and to investigate new component technologies.

This report presents the following GUPFS project activities during FY 2002: results to date of ongoing evaluations and testing methodologies, vendor contacts and relationships established, alternative technologies being watched, and the near-term and longer-term activities planned. Activities related to planning, organizing, and starting up the project are not covered in this report.

1.3 GUPFS Target Technology Evaluations

In order to successfully introduce a scalable, high-performance, parallel, shared file system and to consolidate storage, it is necessary to determine which file system, SAN fabric, and storage technologies will provide the best performance and reliability in the NERSC production environment. To this end, the GUPFS project is conducting evaluations of all of these component technologies to determine which of the existing and emerging alternative technologies are most suitable to the NERSC environment.

During FY 2002 the GUPFS project began developing the methodologies and mechanisms for conducting evaluations of each of the three technology components. Particular emphasis was placed on the techniques for evaluating the shared file systems, with specific attention being focused on benchmarks for parallel I/O and metadata operations. These areas have very few, if any, existing established benchmarks, but they are vital for shared file system performance and scalability. The selected evaluation methodologies, benchmarking approach, and benchmark codes are discussed in Section 4 of this report, "Current Testing Methodologies."

In order to evaluate the shared file systems with a parallel I/O environment representative of the NERSC production environment, the GUPFS project initiated an outreach program to major NERSC customers to determine what I/O mechanisms they use and what their I/O requirements are. The information obtained from this outreach program is being incorporated into the parallel benchmarks being developed for the file system evaluations. The outreach program, which will continue during FY 2003, is discussed in more detail later in this report.

Using the selected evaluation methodology, existing and developed benchmarks, and the initial testbed system, the GUPFS project evaluated two versions of the Sistina GFS file system with Fibre Channel attached storage. These evaluations, which are discussed in more detail later in this report, were conducted after determining the baseline storage and SAN fabric performance. Determining this baseline performance is an important first step in determining which elements of the file system performance are due to the file system and which are due to hardware performance limitations.

1.3.1 Evaluation Goals

In order to assess the characteristics, performance, and suitability of the existing and emerging shared file systems, SAN fabrics, and storage devices, it is necessary to evaluate each of them in the context of the NERSC production environment and parallel jobs. This is particularly true of the shared file system technologies, which are generally immature and vary substantially in design.

In developing the evaluation strategy and methodologies, the goal has been to test and evaluate both the individual components and the collective environment in order to determine the following:

- The correctness of the file system operations and data transfers, down to the storage device level, and as visible to all systems.
- The fault tolerance of all components and the ability of the file system to survive failures of member systems, special servers, network paths, and storage devices for both short-term

and extended periods. This includes the ability to return the file system to a consistent state and to restore it from backups.
- The reliability of all components, particularly the file system, under various degrees, types, and durations of loads.
- The ability of the components to survive extensive stress testing, particularly in the realms of extreme loads and unusual loads, especially in the area of metadata operations.
- The interoperability of the components, particularly the hardware components, in a highly heterogeneous multi-vendor, multiple-fabric, and multi-product environment. The ability of the file systems to operate in such a mixed environment is very important to the ultimate success of the GUPFS project.
- The performance of the file system and hardware components, both individually and in the aggregate, under various loads. Performance is examined on the basis of bandwidth and latency.
- The high-level functionality factors provided by the file system, such as journaling, snapshots, and hierarchical data management interface support.

By exploring the above factors as part of the evaluations, it is expected that the best and most appropriate file systems, SAN fabric technologies, and storage technologies can be identified to ensure a successful GUPFS deployment. It is also expected that any deficiencies identified during the evaluation of these factors will be communicated to the appropriate vendors so that they can be fixed.

1.3.2 NERSC User I/O Profile

Although file system stability and performance benchmarks are important for evaluating and differentiating the potential file systems, it is also important to understand how the file systems will be used by NERSC clients. To this end, the GUPFS project is conducting an informal survey of various NERSC principal investigators to understand how they perform I/O within their large applications. In particular, we are interested in:

- **How the application is run.** This includes the number of processors used in typical and large production runs, what types of files are written and read during the course of the run, what portion of the run time is consumed by I/O, etc. For example, some applications cannot complete during the time limits imposed by the batch queues, and the applications write intermediate checkpoint files to scratch space that can be read by the next batch job in the sequence to continue the calculation.

- **How the I/O is performed in the application.** Does it use an I/O library, such as MPI-I/O (Message Passing Interface-I/O) or HDF (Hierarchical Data Format)? Is the I/O performed by a single process or by multiple processes in parallel? If parallel, do all the processes write to a single file or to separate files? How many files are created within a single run and how large are they? How are they organized and what file system do they write to (scratch, home, etc.)?

- **What is the life cycle of the files read and written by the application?** Which of them get read from and written to the High Performance Storage System (HPSS), and which are discarded after the code completes? Are the files collected into an archive (such as a tar or

cpio file) before being stored to HPSS? Are the files shipped offsite, and if so to where? How is the post-run analysis performed and where?

The information we gain through this survey will help determine which features of a parallel file system are important to the users. For example, parallel file creation by many processes simultaneously within a single directory may perform poorly in a parallel file system due to directory lock contention. On the other hand, this feature may not be as important to the user community in comparison to other features. In addition, we will use this information to develop several benchmarks that emulate the I/O behavior of real NERSC applications.

The NERSC projects are selected by searching the Energy Research Computing Allocation Process (ERCAP) database for principal investigators with large NERSC massively parallel processing (MPP) and storage resource unit (SRU) allocations. We intend to interview between 20 and 30 project principal investigators (PIs) during the course of this investigation. Our procedure is to read the project description from the ERCAP request, contact and interview the PI or technical proxy, and write a summary of the interview. The summary is mailed to the PI for review and corrections. The process is time consuming, taking four or more hours per project. We began the interviews in early October 2002 and have completed three project interviews to date.

In addition to surveying NERSC PIs to understand how their applications perform I/O, the GUPFS project will also use data available in *DOE Greenbook — Needs and Directions in High-Performance Computing for the Office of Science* to further understand the application I/O characteristics and requirements. We will also use an I/O benchmark that will emulate the I/O behavior of NWChem; this benchmark is being developed by a student of NERSC PI Ricky Kendall at Ames Laboratory/Iowa State University.

2 GUPFS Target Technologies

The ongoing evaluations conducted by the GUPFS project during FY 2002 included investigations in all three technology areas that are key to the successful deployment of a center-wide shared file system utilizing consolidated storage resources:

- shared/cluster file systems
- storage devices
- SAN fabrics.

The technologies tested in each of these areas will be discussed in Sections 2.1 through 2.3. In addition to the technologies that were actually tested, alternative and emerging technologies that were investigated and tracked throughout the year will also be discussed.

2.1 File System Technologies

The key technology for the GUPFS project is the shared file system. Without a reliable, scalable, high performance, shared file system, it will be impossible to deploy a center-wide file system. Currently, there are two major technologies for sharing storage between systems: network attached storage (NAS) and storage area network (SAN) [2].

Network attached storage is a general term for storage that is accessible to client systems over general-purpose Internet Protocol (IP) networks from the local storage of network-attached servers. Data transfers between storage servers and clients are performed over a network. This results in the file systems being limited by network bandwidth, network protocol overhead, the number of copy operations associated with network transfers, and the scalability of each server. The file system performance is often limited by the bandwidth of the underlying IP network. The most common example of a NAS file system is the Network File System (NFS). NFS exhibits many of the previously mentioned performance limitations.

Storage area networks, by providing a high performance network fabric oriented toward storage device transfer protocols, allow direct physical data transfers between hosts and storage devices, as though the storage devices were local to each host. Currently, SANs are implemented using Fibre Channel (FC) [3,4] protocol-based high performance networks employing a switched any-to-any fabric. Emerging alternative SAN protocols, such as iSCSI [5], FCIP [6], and SRP [7], are enabling the use of alternative fabric technologies, such as Gigabit Ethernet and the emerging InfiniBand, as SAN fabrics. Regardless of the specific underlying fabric and protocol, SANs allow hosts connected to the fabric to directly access and share the same physical storage devices. This permits high performance, high bandwidth, and low latency access to shared storage. A shared-disk file system will be able to take full advantage of the capabilities provided by the SAN technology.

Sharing file systems between multiple systems is a very difficult problem because file system coherency[2] must be maintained by synchronizing file system metadata operations and coordinating

[2] File system coherency is the correctness, integrity, and consistency of the data and meta-data structures that comprise the file system.

data access and modification. Maintaining file system coherency becomes very challenging when multiple independent systems are accessing the same file system and physical devices.

Shared file systems that directly access data on shared physical devices through a SAN are commonly categorized as being either asymmetric or symmetric. Asymmetric shared file systems allow systems to share and directly access data but not metadata. In asymmetric systems, the metadata is maintained by a centralized server that provides synchronization services for all clients accessing the file system. Symmetric shared file systems share and directly access both data and metadata. Coherency of data and metadata is maintained through global locks, and distributed lock management is performed directly between participating systems.

Shared file systems are not common because implementing them is inherently difficult [8]. Of the two types, asymmetric shared file systems are used more often because centralized metadata servers are easier to implement than distributed metadata management. However, symmetric shared file systems promise better scalability without the bottleneck and single point of failure problems inherent in asymmetric systems.

2.1.1 File System Technology Arena Developments During FY 2002

Over the last year, shared file system technology has been undergoing turbulent changes. Some once-promising shared file systems seem to be fading; other existing shared file systems are continuing to make incremental progress; many new shared file system vendors and alternative technology approaches are appearing; and a major shared file system development project was awarded.

The Sistina GFS file system [9] continues to make limited advances, but the commercial market segment to which it is targeted is becoming more and more remote from scientific HPC. While performance is improving and needed features are slowly appearing, the prospects for Sistina's long term survival are questionable. Experience has shown that the promising Device Memory Export Protocol (DMEP), which Sistina was championing as a means of scaling and accelerating distributed locking, does not perform as well as IP-based lock managers. Sistina is now switching to a new IP-based lock manager and is strongly considering dropping support for DMEP in future releases. This essentially dooms the attempt to make DMEP part of the SCSI standard, and the few storage vendors that have supported DMEP are also looking at dropping DMEP support. Worst of all from the GUPFS perspective is that Sistina is not planning to work on scaling beyond 64 or 128 nodes because of their low-end market focus.

On a more positive note, ADIC is continuing to develop their CentraVision File System (CVFS), which will soon to be renamed StorNext [10]. ADIC is interested in working with NERSC and other DOE laboratories to make CVFS work in the HPC environment and to integrate it with HPSS. The GUPFS project currently has CVFS installed and is awaiting the execution of an evaluation agreement before beginning testing.

IBM's General Parallel File System (GPFS) continues to advance, both on large SP systems and on IBM Linux clusters. In addition to being used on NERSC's SP system, GPFS is installed and being tested on the Berkeley Lab Alvarez Linux cluster. Unfortunately, licensing issues and limited

platform and operating system support under IBM's current marketing plan limit GPFS's usability and appeal. However, NERSC is advocating making GPFS portable to non-IBM hardware.

Many new shared file system vendors are appearing, some with interesting and promising technologies. These include Maximum Throughput with their local area network (LAN)-distributed InfinARRAY file system, Ibrix with their somewhat similar file system, and a host of other vendors including 3PAR, PolyServe, and SANique. It is heartening to see such interest in shared file systems, but this technology arena is now quite turbulent and volatile.

One very positive development in the last year was the Advanced Simulation and Computing (ASCI) PathForward program's award of the Scalable Global Secure File System (SGSFS) [11] development contract to Hewlett-Packard for the development of the Lustre file system [12]. Since SGSFS shares a number of the same endpoint functionality goals as GUPFS, Lustre is likely to be one of the candidate shared file system solutions for GUPFS. However, NERSC also has requirements that are not fully reflected in the ASCI workload, so it remains to be seen if Lustre can satisfy all the requirements of each workload.

The GUPFS project was able to conduct limited, but more detailed, investigations of the Sistina GFS file system during FY 2002. Unfortunately, time pressures resulting from the myriad other activities associated with starting up the GUPFS project limited the time available for this investigation. The GFS file system proved to be an important tool in developing the file system evaluation methodologies, benchmarking approach, and benchmark codes for the GUPFS project.

2.1.2 The GFS File System

Shared file systems can be classified based on a number of their design characteristics. These characteristics include whether the shared file systems are client-server based (traditional NAS) or share physical storage devices (traditional SAN). SAN shared-storage file systems may be further characterized as symmetric or asymmetric based on the way they maintain metadata and data coherency. New LAN-distributed hybrid file systems that utilize SAN-attached storage are also beginning to appear.

The Sistina Global File System (GFS) [13] is a SAN-based, symmetric shared-disk file system. Indeed, it is one of the few symmetric shared file systems in existence. As a symmetric shared file system, GFS offers full access to both the metadata and data from all connected systems. It maintains data and metadata consistency through global locks, and it supports the use of distributed lock management.

GFS has been under development for a number of years, originating at the University of Minnesota in 1995 [14,15]. At the beginning of FY 2002, GFS was under active development by Sistina Software, Inc. as an open source project, targeted to cluster systems employing Fibre Channel SAN-attached storage. GFS is designed to be scalable and to perform nearly as well as local file systems by utilizing the client system's local buffer cache for read and write caching. GFS also supports distributed lock management through hardware using the novel Device Memory Export Protocol (DMEP) SCSI extension.

GFS 4.2

In early October 2001, Sistina released GFS version 4.2. This release had support for advanced DMEP hardware locking, which was just becoming available from a few storage vendors. The GFS 4.2 release also contained a substantial number of bug fixes, as well as a number of early experimental features. Unfortunately, the GFS 4.2 release was no longer open source software, although its source was freely available under the limiting Sistina Public License (SPL).

The GUPFS project obtained the GFS 4.2 software source shortly after it became available. Near the end of October, Sistina changed its licensing policy and made GFS a proprietary, closed product, subsequently only available for purchase. Fortunately, the GUPFS project had already obtained the source under the original terms of the SPL, and could still use it at no cost.

The GUPFS progenitor project was specifically targeted towards investigating GFS as a SAN-based shared-disk file system in the scientific cluster environment and had no budget to buy file system software, because at that time GFS was free, open source software. Therefore, the GUPFS project continued to use the already obtained GFS 4.2 software as the basis of much of its methodology development, benchmark code development, and evaluation activities throughout FY 2002.

The performance of GFS 4.2 file I/O and metadata operations were evaluated, and the raw performance of the underlying GFS 4.2 pool drivers was also evaluated to differentiate the actual GFS file system performance from the performance of the GFS pool logical volume manager. The results of these evaluations are presented in Section 5 of this report and in Appendix B.

GFS 5.1

As part of the GUPFS testbed technology update, the GUPFS project received funding to purchase a 25-node GFS license, which will enable scaling tests and will also allow some licenses to be used periodically in conjunction with other NERSC Linux systems for usability testing. The GFS 5.0 license and support contract were acquired in June 2002. Shortly thereafter, the GFS 5.1 maintenance release was issued and obtained under the support contract.

In addition to many bug fixes, GFS 5.0 provided the following advancements:

- Improved file system performance.
- A new, vastly improved IP-based lock manager.
- Support for standard RedHat and Suse Linux distributions.
- A new global network block device (GNBD) facility with greatly improved performance.
- Initial implementations of file system quotas, file locking, and Direct I/O.

The GFS 5.1 release provided additional bug fixes and the following advancements:

- Support for newer stock Linux kernels, and newer RedHat and Suse Linux distributions and kernels.
- Improved and functional Direct I/O.
- Enhanced file system tuning functionality.
- Application-specific file system tuning capabilities.
- File system object attribute lists.

GFS 5.1 was installed on the new nodes of the updated GUPFS testbed in September 2002 with onsite assistance from Sistina support personnel. Initial GFS file systems were installed and lightly tested utilizing Sistina's new IP-based Global Universal Lock Manager (GULM). An attempt was then made to install a GFS file system utilizing hardware DMEP locking in order to conduct direct performance comparisons with GFS 4.2. This attempt was initially stymied by the failure, in rapid succession, of both of the primary DMEP-capable storage devices (DotHill and Silicon Gear).

Once one of the DMEP capable storage devices (the Silicon Gear Mercury II) was repaired, activity resumed on setting up a GFS 5.1 file system using hardware DMEP locking. Problems were encountered in setting up the DMEP hardware lock tables on the storage device for use by the file system locking modules. A series of two binary patches from Sistina succeeded in resolving this problem. After basic GFS 5.1 functionality was verified, the performance of GFS 5.1 file system file I/O and metadata operations were evaluated, along with the raw performance of the underlying GFS 5.1 pool drivers. The GFS pool drivers provide primitive logical volume management functionality that supports striping across multiple storage devices. The raw performance of the underlying GFS pool devices was evaluated to quantify the performance of the pool driver in relation to the raw storage performance, in order to differentiate between the performance of the GFS pool logical volume manager and the actual GFS file system performance. The results of these evaluations and a comparison with GFS 4.2 performance are presented in Section 5 of this report and in Appendix B.

2.2 Storage Technologies

The GUPFS project evaluation phase targeted three technology areas to evaluate/investigate: storage, SAN fabric, and file system. A new category of storage systems called *utility storage* is currently being introduced by many storage startups such as 3PRAdata and YottaYotta. Through innovative hardware and software design, these storage startups have promised to deliver a higher level of scalability, connectivity, and performance. For example, both YottaYotta's NetStorager and 3PARdata's InServ storage systems use parallel computing and clustering technology to provide increased connectivity and performance scalability, and redundancy for higher reliability.

The building blocks of utility storage can scale from a few dozen to a few hundred terabytes, or even in principle to the petabyte range, with a transfer rate as fast as a few thousand megabytes per second and the ability to sustain beyond 100,000 I/O operations per second (IOPS) for online transaction processing (OLTP) applications. To save floor space, these storage vendors have adopted a modular design to achieve an exceptional storage density. With the built-in redundancy in the architecture, any components can fail-over to one another, and each component can be hot-swapped without affecting other activities.

Since May 2002, we have been beta testing YottaYotta's NetStorager system. The evaluation has been very successful. The NetStorager has improved substantially and is now delivering very good scalable performance.

A major objective of the GUPFS project for FY 2003 is to be able to demonstrate sustained file system performance of 1 gigabyte per second (GB/s), which equals 8 gigabits per second (Gb/s). The 8 Gb/s goal demonstrates that a global parallel file system is capable of passing the 2 Gb/s

performance bar that is currently set by NFS/NAS servers. The existing storage on the GUPFS testbed does not have adequate aggregate bandwidth to provide 1 GB/s sustained file system performance. Testing during FY 2002 has demonstrated that the current storage provides only a fifth of the required aggregate bandwidth, even when used as dedicated local storage assigned to separate compute nodes. This inadequate bandwidth is due to the 1 Gb/s Fibre Channel interfaces supported by the existing storage, the limited number of ports on the controllers, and limitations in controller performance.

While storage startups are introducing new storage technologies, the existing storage vendors are also introducing storage systems supporting 2 Gb/s FC ports and an increased number of ports for SAN connectivity. One such example is EMC's CX 600 storage. The CX 600 supports up to eight 2 Gb/s FC ports with a possible 16 Gb/s peak aggregate performance.

One of the major storage issues encountered during the FY 2002 evaluation is scalability—how well the storage is able to process simultaneous I/O requests from multiple clients (initiators) to a single device or *logical unit number* (LUN). The envisioned GUPFS system is a *shared-disk* file system with a large number of clients (10,000s of CPUs). The storage systems selected for the GUPFS project have to be able to handle the I/O requests from this large number of clients to the shared devices very efficiently. Our evaluation found that scalability of single-device access is a common problem in many storage systems, in terms of both the number of initiators (clients) allowed on a single device and the performance of shared access. On most storage systems, the maximum number of client initiators allowed on a single LUN has been less than 256, or even fewer. Shared-access performance has also been very poor on many storage systems when the number of clients increases. Although we believe that the storage vendors will address these issues before GUPFS is ready for deployment, we will continue monitoring developments in this area and work with the vendors to steer their development effort to support the needed functionalities. We have already been successful in getting YottaYotta to address these issues.

2.3 SAN Fabric Technologies

An important goal of the GUPFS project is to evaluate the performance characteristics of individual components and the interoperability of these components in a heterogeneous multi-vendor and multiple-fabric environment. The ability of the file systems to operate in such a mixed environment is very important to the ultimate success of the GUPFS project.

2.3.1 2 Gb Fibre Channel Switch

Fibre Channel technology has recently been upgraded from 1 Gb/s to 2 Gb/s bandwidth. This increase will allow substantially improved storage performance to be realized. We have included two additional switches to the test bed:

- one Brocade SilkWorm 3800 16-port 2 Gb Fibre Channel switch
- one Qlogic SANbox2-16 16-port 2 Gb Fibre Channel switch.

It is important to investigate the improved Fibre Channel technology and determine what level of storage performance can be achieved.

2.3.2 iSCSI Switch

Additional SAN Fabric technologies are beginning to appear. Using Ethernet as a SAN fabric is now becoming possible due to the iSCSI standard, which is a block storage transport protocol. The iSCSI protocol allows the standard SCSI packets to be enveloped in IP packets and transported over standard Ethernet infrastructure, which allows SANs to be deployed on IP networks.

This option is very attractive as it allows lower-cost SAN connectivity than can be achieved with Fibre Channel, although with lower performance. It will allow large numbers of inexpensive systems to be connected to the SAN and use the cluster file system through commodity-priced components. While attractive from a hardware cost perspective, this option does incur a performance impact on each host due to increased traffic through the host's IP stack.

Figure 1 depicts how iSCSI can be used for storage access.

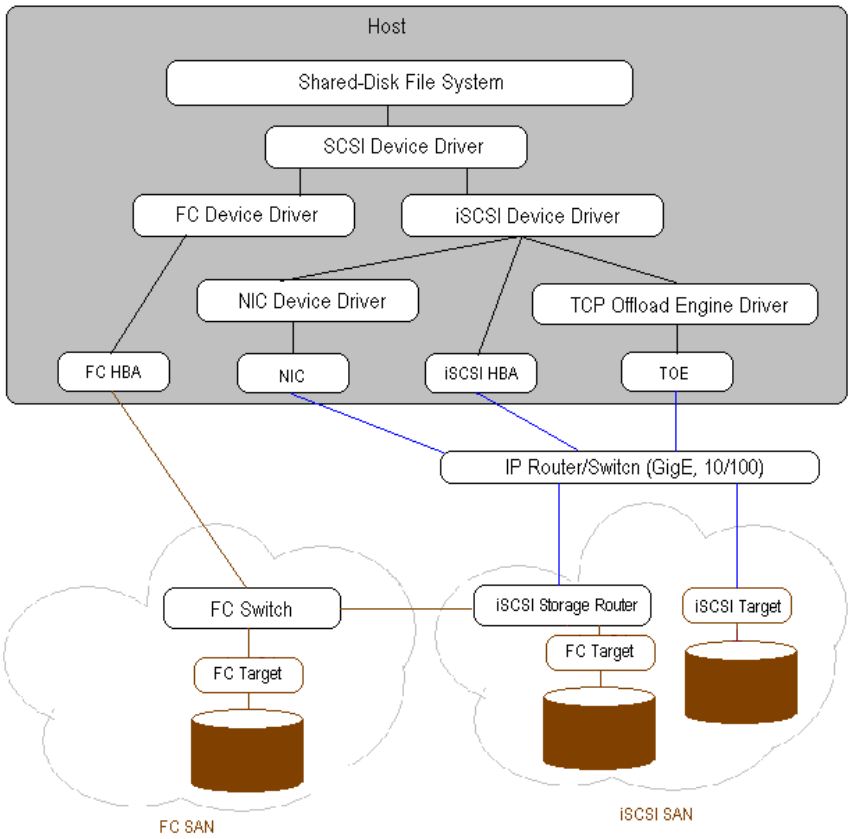

Figure 1. Mechanisms for accessing storage from a host using iSCSI.

Figure 1 shows that on the host side, there are four different I/O paths: Fibre Channel (FC), iSCSI over traditional Gigabit Ethernet (GigE) or 10/100 network interface card (NIC), iSCSI over TCP (Transmission Control Protocol) Offload Engine (TOE), and iSCSI over iSCSI HBA (host bus adapters, e.g., Intel IP Pro/1000 T IP HBA). On the storage side, the diagram also shows that the iSCSI traffic can go to an iSCSI storage target directly through an iSCSI storage router (e.g., Cisco's SN5428 router) to any FC storage, or through an iSCSI storage router connected to a FC

switch (e.g., Brocade SilkWorm 3800 or Qlogic SANbox2) to any FC storage. All these I/O paths need to be evaluated for their appropriateness for GUPFS based on their price-performance characteristics, scalability, fabric management, etc.

An important part of the iSCSI technology that also needs to be examined is fabric bridges between Ethernet networks and Fibre Channel SANs.

2.3.3 InfiniBand Switch

In addition to using Ethernet as a SAN, the newly emerging InfiniBand interconnect also shows promise for transporting storage traffic in a SAN. InfiniBand offers performance (bandwidth and latency) beyond that of either Ethernet or Fibre Channel, with even higher bandwidths planned.

Figure 2 shows how a single InfiniBand (IB) interface on a host can be used for both IP traffic and FC traffic to perform I/O. On the host side, I/O can be accomplished through the IB Fibre Channel driver to access any FC SAN, or through iSCSI over the IB NIC driver to access any iSCSI SAN or FC SAN.

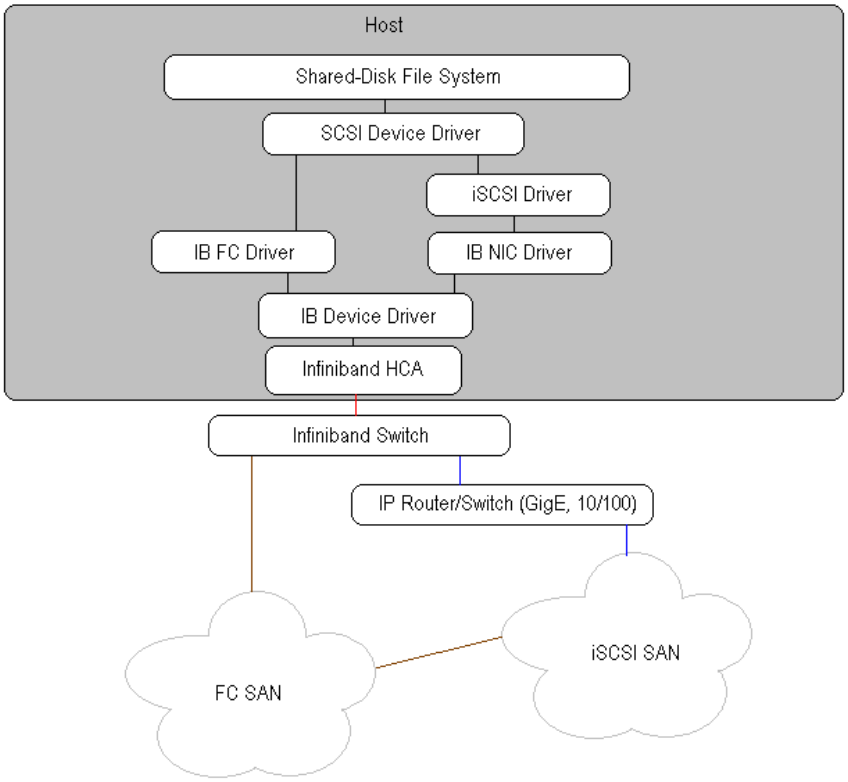

Figure 2. How a host can access both IP- and FC-based SANs with a single InfiniBand interface.

With the promise of commodity-level pricing in the future, InfiniBand is a technology that needs to be studied, even in its very early stages. As with Ethernet fabrics, an important part of the InfiniBand technology that needs to be examined is fabric bridges between InfiniBand and Fibre

Channel SANs. Another area of InfiniBand technology that needs to be explored is storage transfer protocols (e.g., SCSI Remote-DMA Protocol or SRP) and methodologies.

2.3.4 High Speed Interconnect

With the increased interest in large-scale shared file systems, several promising shared file system architectures have appeared. Many of these new file system architectures are designed for the high performance cluster environments, typically with some kind of high speed interconnect for the messaging traffic. Many of the new architectures perform storage transfers between client nodes and storage nodes over the high speed interconnect.

Storage virtualization has been widely used as a way to provide higher capacity or better performance. Virtualization can also be implemented at the file system level. File system virtualization allows multiple file systems to be aggregated into one single, large virtual file system to deliver higher performance than a single NFS or NAS server could do. A few examples of federated file systems include HP's DiFFS, Max-Throughput's InfinARRAY, Ibrix's SAN-based file system, etc.

Figure 3 shows a possible architecture for the federated file systems using the high speed interconnect for the storage traffic.

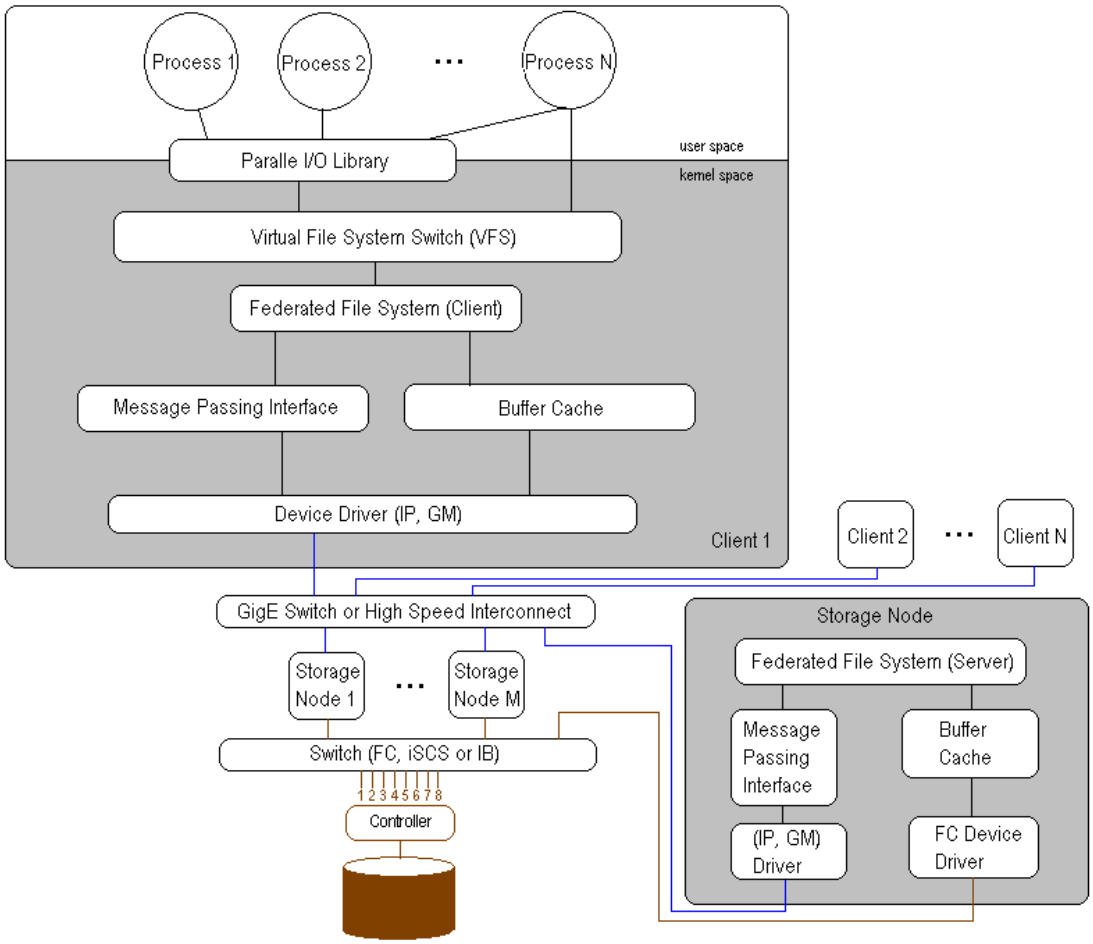

Figure 3. Possible federated file system architecture.

3 Testbed Configuration

During the first year of the GUPFS project, the testbed system used for conducting evaluations and gaining familiarity with the component technologies was the testbed originally constructed for the GUPFS progenitor project. That smaller project was to investigate the suitability of shared-disk file systems in a SAN environment for scientific clusters. This initial testbed system was used to develop the evaluation methodologies and techniques to be used in the evaluation phase of the GUPFS project. It was also used as a platform for developing the GUPFS benchmark approach, examining the suitability of existing benchmark codes and the development of the evolving parallel file system benchmark codes being written for the GUPFS project. Although small, this testbed was adequate for conducting initial studies of the Sistina GFS file system, and was a useful resource in attracting the attention of component technology vendors and developing relationships with a number of these vendors.

With the advent of the NERSC Strategic Proposal and the evolution of the GUPFS project within that proposal, the scope of the project and the component technologies that needed to be investigated expanded greatly. The broader scope of the project, the cluster file system, SAN fabric, and storage technologies needing to be evaluated, and the extremely rapid evolution of these component technologies quickly made it clear that the initial testbed system was inadequate. To remedy this inadequacy, an expansion of the testbed system was designed that would provide sufficient hardware and computational resources to support the evaluation of multiple new component technologies. The expanded testbed was designed to provide underlying SAN fabric and storage resources with sufficient aggregate performance to stress-test existing and emerging shared file system technologies. This design emphasized the extensibility of the testbed system in order to accommodate future technology developments.

The expanded testbed system design was finalized at the end of the third quarter of FY 2002. The components were identified, tested, and procured during the fourth quarter, and the components were assembled and integrated with the existing testbed system at the end of FY 2002. The configuration of the expanded testbed system is discussed in detail later in this section.

3.1 Initial Testbed Configuration

The initial testbed was a small-scale system mimicking a scientific parallel computational cluster that was constructed from commodity components. Its original purpose was to assess the suitability of shared-disk cluster file systems with SAN attached storage to the scientific computational cluster environment.

This testbed system presented a microcosm of a parallel scientific cluster—dedicated computational nodes, special-function service nodes, and a high-speed interconnect for message passing. It consisted of five computational nodes and one management/interactive node, and utilized an internal jumbo frame Gigabit Ethernet as the high-speed message passing interconnect. An internal 10/100 Mb/s Fast Ethernet LAN was used for system management and NFS distribution of the user home file systems. The configuration of the testbed is illustrated in Figure 4.

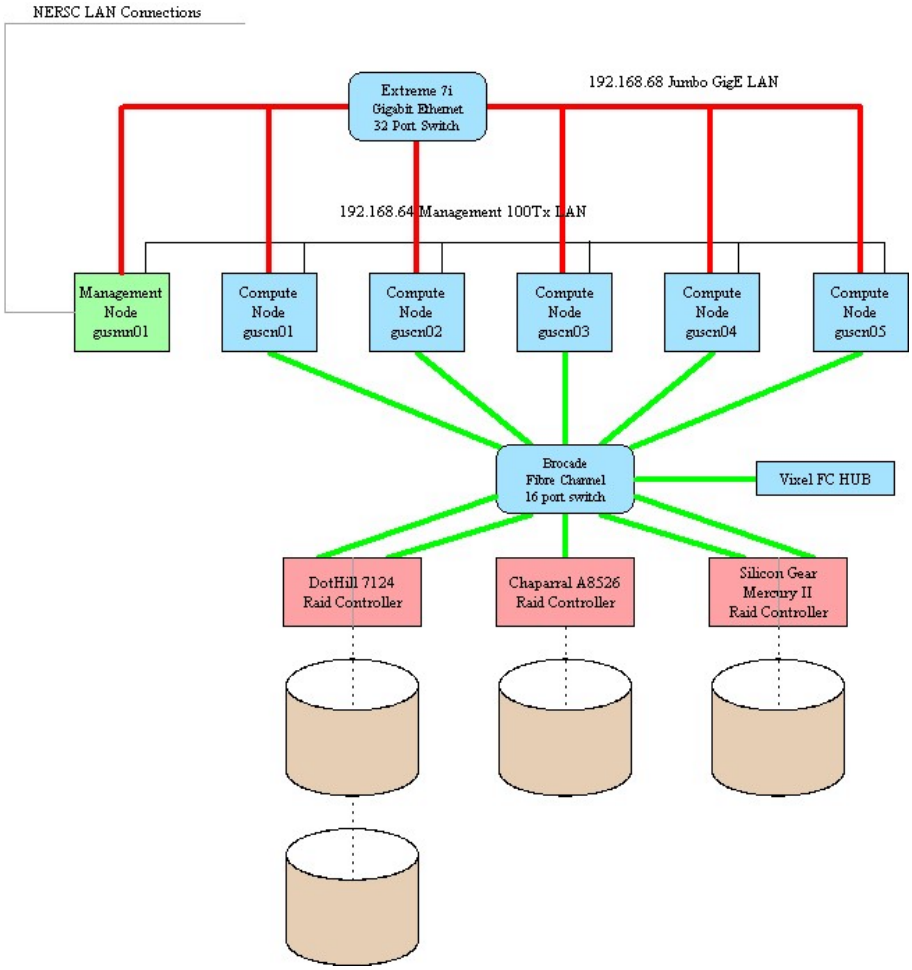

Figure 4. Initial GUPFS testbed configuration.

All six nodes of the initial testbed were identical Pentium III systems with dual 1 GHz CPUs, 1 GB of PC133 ECC (error-correcting code) memory, a single 18 GB 10,000 RPM SCSI drive for local storage, and motherboards with two 64-bit 66 MHz PCI (peripheral component interconnect) slots. The management/interactive node and the computational nodes differed little in configuration. All nodes contained a Gigabit Ethernet card supporting jumbo frames. The computational nodes were equipped with a 1 Gb/s Fibre Channel interface card, while the management node contained additional Fast Ethernet cards and an additional Gigabit Ethernet card.

All nodes were installed in 4U rack mount cases. The basic configuration of all nodes was the same:

- Intel Server Board STL2 motherboards, with two 64-bit 66 MHz PCI slots with additional 32-bit PCI slots
- dual Pentium III 1 GHz CPUs
- 1 GB of PC133 ECC memory
- on-board VGA (video graphics array) graphics
- on-board U160 Adaptec SCSI controllers

- one on-board Intel PRO/100 Ethernet interface
- one 18 GB Ultra 160 LVD 10k RPM SCSI disk drive
- one Qlogic qla2200 64-bit Fibre Channel HBA with copper HSSDC (high speed serial data connector) (compute nodes only)
- one SysKonnect SK-9821 64-bit PCI Gigabit Ethernet NIC (the management node had two).

All testbed nodes ran Linux, based on the RedHat 7.1 distribution, using custom-built Linux 2.4.9 kernels containing GFS 4.2 support patches. The testbed supported parallel job submission and execution using Open PBS and utilized MPICH as the MPI implementation for parallel jobs. Portland Group C, C++, and various flavors of FORTRAN compilers provided the compilation and execution environment for the parallel jobs.

Each computational node was connected to a switched 1 Gb/s Fibre Channel SAN fabric by a Qlogic 2200 FC Host Bus Adapter (HBA). The switched FC SAN fabric utilized copper FC cables connecting the nodes and storage to a 16-port Brocade Silkworm 2800 FC switch. Through the SAN fabric, the computational nodes were provided access to three different storage devices. These storage devices, all of which were RAID storage subsystems, were selected on the basis of their support for the Device Memory Export Protocol (DMEP) SCSI standards extension. This was of interest at that time because the testbed was initially targeted toward early evaluations of the Sistina GFS file system, which at that time utilized the novel and promising DMEP in its distributed lock management.

The disk storage devices connected to the Fibre Channel SAN fabric were:

- a dual-controller DotHill 7124 RAID subsystem, with an expansion cabinet
- a dual-controller Silicon Gear Mercury II RAID subsystem
- a single-controller Chaparral A8526 RAID subsystem with attached storage.

Each of the RAID controllers had dual Fibre Channel ports for connecting to the switch, both of which could be used simultaneously. All three storage devices supported RAID 0, 1, 3, and 5 configurations and utilized similar 10,000 RPM 73 GB U160 SCSI or FC-AL disk drives. The DotHill contained 20 drives, the Silicon Gear 12 drives, and the Chaparral 10 drives. Total unformatted SAN attached storage capacity was approximately 3.1 TB, with a nominal maximum of 2.7 TB of formatted RAID 5 storage.

In addition to being selected for exploring the use of hardware DMEP with GFS 4.2, the storage configuration was chosen to enable the exploration of the relative performance, reliability, and interoperability of multiple storage vendors' products. The quantity and character of the storage was dictated by:

- the desire to be able to achieve maximum performance from each storage controller
- the desire to be able to explore the Linux support for file systems greater than 2 TB on 32-bit architectures when such support became available
- price.

The initial testbed system provided a useful facility for developing the benchmark methodology and special benchmark codes for the GUPFS project. It was also useful in helping to establish the

credibility of GUPFS with technology vendors, and in building relationships with various technology vendors. However, with the expanded scope of the GUPFS project and the inexorable progression of technical advancements, it became apparent that the initial testbed was inadequate for conducting the types and levels of technology evaluations needed for the first phase of the GUPFS project.

3.2 Updated Testbed Configuration for FY 2003

The initial testbed system, designed for a much more limited project, proved to be inadequate for the broader scope of the GUPFS project. Technological advancements over the last year have outstripped the ability of the initial testbed to incorporate them. Experience with the testbed and attempts to integrate new storage and fabric technology into it demonstrated that the testbed was not flexible enough to allow emerging advanced technologies to be integrated into it for evaluation, even in the near term. Given the technologies that the GUPFS project plans to begin evaluating in FY 2003, it was clear that the existing testbed could not accommodate their inclusion.

3.2.1 Updated Testbed Design Considerations

In designing an updated testbed for the GUPFS project, a number of factors were considered. These included the new requirements for the expanded scope and goals of the GUPFS project, lessons learned from using the initial testbed, the new and emerging technologies needing to be investigated and evaluated over the next several years, limitations encountered in the initial testbed, and the near-term investigations that are planned.

A number of lessons were learned from using the initial testbed during the first year of the GUPFS project. These lessons directly affected the design of the upgraded testbed. The primary lessons learned were:

- **More 64-bit PCI slots are needed in the nodes.** Each of the initial testbed nodes contained two 64-bit, 66 MHz PCI slots. At the time the initial testbed was designed and acquired, nodes with 64-bit PCI slots were rare. The motherboard that was used was explicitly selected because it had two 64-bit PCI slots. Even when the initial testbed was designed, higher performance I/O devices (Fibre Channel and Gigabit Ethernet) were starting to be built as 64-bit PCI cards in order to increased performance and memory bandwidth. This is even truer today. Almost all high performance I/O devices now require 64-bit PCI slots, and many are now supporting PCI-X. Only having two 64-bit PCI slots, which are consumed by the base FC HBA and Gigabit Ethernet card, prevents installing additional high performance I/O devices, such as Myrinet interfaces, InfiniBand HCAs, and iSCSI HBA/NIC cards. Two slots severely limits the flexibility of the testbed and the ability to explore the new fabric technologies. This limitation led to the selection of a new motherboard with six 64-bit PCI/PCI-X slots.

- **More special-purpose nodes are needed.** The original testbed only had one special-purpose node—the management node. All other nodes were compute nodes. Experience showed that a number of other special-purpose nodes were needed to conduct activities without impacting the availability of the compute nodes for benchmark runs. Dedicated special-purpose nodes were needed for code development and benchmark testing, as storage

servers for NFS gateway functions, and as installation targets for preparing and testing new kickstart installations using Intel's Pre-Execution Environment (PXE) network bootstrap for the OS versions needed by the various file system and fabric interfaces being tested. Without adequate nodes to dedicate to special purposes, it was necessary to use compute nodes for these purposes, which adversely impacted the availability of nodes to conduct benchmark runs.

- **Don't use copper Fibre Channel cables.** While copper HSSDC Fibre Channel cables are cheaper than optical cables, they are fragile, bulky, and too stiff. A number of the copper HSSDC cables failed with broken connector pins during the first year. Also, the large number of cables needed to connect all the storage and nodes to the SAN fabric made it difficult to find space to run the cable bundles and to provide strain relief, which added to the difficulty of changing the configuration of the fabric. These factors contributed to the decision to use optical cables for all new equipment.

- **Larger local disks are needed.** This is not because of heavy use of local storage by benchmark runs. Rather, it arises from the desirability of installing multiple, completely independent versions of Linux into separate partitions on the local disks. Doing this facilitates quickly switching between the various Linux kernels and distributions required to test specific file systems, fabric interfaces, and different versions of the file system. Being able to quickly boot very different software environments allows the GUPFS project members to evaluate multiple file systems, storage devices, and fabric hardware in a timely manner.

- **More spare interface cards are needed.** This is not because of component failures, though those do occur. This is necessary to be able to take advantage of unexpected opportunities and unforeseen avenues of investigation. It is also necessary to have additional spare cards in order to conduct cooperative investigations involving other systems outside of the GUPFS testbed (such as PDSF and Alvarez).

In addition to incorporating the relevant lessons learned from the original testbed, the design of the upgraded testbed was also influenced by limitations that were encountered on the original testbed. These limitations included the following:

- **The Fibre Channel fabric did not have enough switch ports.** The combination of node and storage controller connections to the FC switch left only two of the 16 ports free. The connection between the Vixel hub and the Brocade switch used one of these free ports, leaving only one port available. The newer storage devices being acquired or tested have multiple fabric connections from a single controller. The NetStorager currently has eight fabric connections, and the newly acquired EMC CX600 also has eight fabric connections. In order to test the full performance of these storage devices, it was necessary to disconnect the other storage, which negatively impacted file system testing.

- **The original 1 Gb/s storage devices do not have enough aggregate bandwidth to truly stress-test the shared file systems being evaluated.** This is particularly true when conducting scalability tests using multiple nodes. This is caused partially by the limited

number of fabric port connections provided by these storage devices, partially by design limitations, and partially due to the 1 Gb/s link speed, which contributed to the storage controller design limitations.

- **The testbed Gigabit Ethernet switch does not have enough switch ports for complex configurations and connection to the NERSC network.** The introduction of additional nodes, Gigabit Ethernet fabric bridges, and iSCSI HBA offload cards will exceed the port capacity of the original 32-port switch. To facilitate testing with other systems, it is desirable to connect the GUPFS testbed Gigabit Ethernet switch to the NERSC floor networks as a restricted subnet, and to directly connect with the switches in other systems. There are not enough ports to do this with the existing switch, particularly if multiple links are bonded together for increased uplink bandwidth.

- **The original GUPFS testbed SAN fabric is too simple to adequately study SAN fabric performance and interoperability issues.** The original fabric consists of a single Fibre Channel switch and a single FC hub. This fabric does not allow for the study of trunking between switches, multiple-switch transfer latencies, and multiple-vendor equipment interoperability.

In addition to incorporating the relevant lessons learned and correcting the observed limitations of the original testbed, the design of the upgrade was strongly influenced by the interesting new and emerging technologies that the GUPFS project plans to investigate in the near future: iSCSI for doing storage transfers over Ethernet, InfiniBand for doing storage transfers via SRP and for IP traffic, 2 Gb/s Fibre Channel, and the Lustre and InfinARRAY shared file systems. Each of these technologies has direct and indirect ramifications that impacted the design of the testbed upgrade.

The near-term investigations that the GUPFS project plans to conduct using the upgraded testbed system include:

- **Evaluating the iSCSI protocol for accessing storage using Ethernet networks.** The final iSCSI standard has been completed and is awaiting formal adoption by the standards committee. The GUPFS project intends to evaluate both purely software iSCSI implementations that use standard Ethernet NICs and the specially designed iSCSI HBA offload card hardware. This will be done to determine the relative performance and the merits of each of these approaches.

- **Evaluating the InfiniBand interconnect as a SAN fabric using the SCSI Remote-DMA Protocol (SRP).** InfiniBand is a high performance interconnect that shows promise as both a cluster message passing interconnect and as a high performance SAN interconnect. The feasibility and performance of iSCSI over InfiniBand will also be investigated.

- **Evaluating 2 Gb/s Fibre Channel.** This investigation will include the evaluation of 2 Gb/s HBAs, multiple 2 Gb/s switches from multiple vendors, and 2 Gb/s storage devices. Complex FC fabrics, including connections through multiple switches, will be studied. The existing 1 Gb/s FC fabric and storage devices will be integrated with the 2 Gb/s fabric to

study the issues of interoperability and performance impacts. The issue of interoperability between multiple vendors in a single fabric will also be investigated.

- **Evaluating the ADIC StorNext (CVFS) shared file system.**

- **Evaluating Maximum Throughput's InfinARRAY shared file system.**

- **Evaluating the Lustre object-oriented shared file systems.**

- **Attempting to demonstrate 1 GB/s aggregate sustained I/O transfers with the various shared file systems.**

- **Evaluating the bridging of multiple SAN fabrics together into a single SAN.** A single SAN fabric incorporating Fibre Channel (1 and 2 Gb/s), Gigabit Ethernet, and 1x and later 4x InfiniBand will be constructed and tested. This will be done using Gigabit Ethernet to Fibre Channel, and InfiniBand to Fibre Channel and Gigabit Ethernet fabric bridges. All three SAN fabrics will be used to simultaneously access the same FC storage devices.

Each of these planned investigations contributes configuration requirements for the testbed upgrade design.

The combination of the emerging technologies and the planned investigations resulted in a number of specific requirements for the design of the GUPFS testbed. Some of these requirements included:

- **The new fabric technologies all require 64-bit PCI slots in order to achieve reasonable performance.** The 2 Gb/s FC HBAs require 64-bit PCI slots, as do the InfiniBand HCAs, the Gigabit Ethernet NICs, the Myrinet interfaces, and the iSCSI HBA/NIC cards. The FC HBAs and Gigabit Ethernet NICs are capable of operating with reduced performance in 32-bit PCI slots. The InfiniBand HCAs, Myrinet interfaces, and iSCSI offload cards can only use 64-bit PCI slots. This, in conjunction with the need to put more than one of these fabric interfaces into each node to make testing easier, drove the selection of node motherboards with a high number 64-bit PCI slots.

- **Many PCI cards for the emerging hardware technologies are being initially produced only in standard height (4U) form factors.** Examples of this include early Gigabit Ethernet cards and the current 1x InfiniBand HCAs and iSCSI HBA/NIC cards. Only later, more advanced versions of these cards will be available in low-profile form factor suitable for 2U node cases. Some older technologies are still only available in standard-height form factors, for example Myrinet interfaces cards, although low-profile versions are on the horizon. Given the situation where most new technologies are initially appearing only on standard-height cards, while some mature technologies still only appear on standard-height cards, it will be necessary for at least some of the new technology nodes to have 4U cases, while the majority of the new nodes can go in 2U cases to save space.

- **PCI buses, even 64-bit, 66MHz PCI buses, do not have enough bandwidth to service the new fabric technologies.** Gigabit Ethernet is at the limit of PCI bus bandwidth. The bi-directional 2 Gb/s Fibre Channel and 1x InfiniBand (2.5 Gb/s) fabrics exceed the bandwidth of even 64-bit, 66 MHz PCI buses. The forthcoming 10 Gb/s Ethernet and 4x InfiniBand (also 10 Gb/s) technologies completely overwhelm all PCI buses. The poor memory transfer latency of PCI also impacts performance. In order to fully realize the performance of the new fabric technologies, it is necessary to use PCI-X buses operating at 100 or 133 MHz. PCI-X buses operating at 133 MHz are capable of 8 Gb/s bandwidth and have substantially improved latency. The need to incorporate and evaluate bidirectional 2 Gb/s and 10 Gb/s technologies necessitated the selection of motherboards with PCI-X buses. Unfortunately, PCI-X is also an emerging technology, making it difficult to find motherboards that support it, particularly at 100 and 133 MHz bus speeds. Given the need to install multiple interface cards in a single system, it was also necessary to find motherboards with more than two PCI-X slots.

- **Given the need for PCI-X buses, it was sensible to get as many of the system component cards as possible in PCI-X versions** in order to obtain maximum performance.

- **The effort to demonstrate sustained file system I/O transfer rates in excess of 1 GB/sec** will require the use of 2 Gb/s Fibre Channel. This in turn requires PCI-X buses and PCI-X cards, providing yet another impetus for PCI-X based systems.

- **The ADIC StorNext file system uses a metadata server to manage data and metadata coherency and synchronization.** Multiple metadata servers can be used to provide high-availability fail-over support and to improve performance with multiple StorNext file systems. The Lustre file system also uses metadata servers, and the new GFS IP-based lock manager GULM needs to run on a node that does not mount the GFS file system. This requires at least one dedicated special-purpose system to run any of these three file systems.

- **The Lustre file system uses object storage targets (OSTs) to serve storage to the Lustre client systems.** OSTs are Linux systems running OST service modules to serve locally or SAN attached storage to the Lustre client systems over IP networks. The Maximum Throughput InfinARRAY file system uses storage processors (SPs) to serve local or SAN attached storage to clients over IP networks. SPs are also Linux systems running special software. In order to evaluate either of these file systems, it will be necessary for the upgraded testbed to have a number of nodes that can be used as storage servers, while still having enough compute nodes configured as clients on which to run benchmarks codes.

- **The InfinARRAY file system plans to provide support for storage transfers over Myrinet interconnects using the Myricom GM protocol.** This promises to provide substantially better performance compared to IP protocol transfers over Gigabit Ethernet. In order to evaluate this functionality, the upgraded testbed will need to in corporate a small Myrinet switch. This leads to the interesting prospect of incorporating the line card from this switch into one of the unused slots in the Berkeley Lab Alvarez cluster Myrinet switch, effectively making the GUPFS testbed SP nodes part of the Alvarez cluster. This would

allow scaling studies of the InfinARRAY file system to be conducted on many more nodes than are available in the GUPFS testbed.

In addition to incorporating all considerations based on the lessons learned from the original testbed, observed limitations of that system, and the near-term investigations and evaluations planned, the design of the updated testbed was driven by three overriding goals:

1. The updated testbed configuration needed to be flexible, expandable, and adaptable to cover shifts in technological trends and allow for unanticipated opportunities and directions over a period of several years.

2. The updated testbed needed to be able to support multiple simultaneous activities, such as two simultaneous independent evaluations during preparation and setup for new evaluations.

3. The updated testbed needed to be able to be temporarily integrated with other systems in order to conduct scalability studies with larger numbers of nodes.

3.2.2 Updated Testbed Configuration

The final configuration of the updated testbed was completed in the beginning of the fourth quarter of FY 2002. It was based on all of the considerations discussed in the previous section. This configuration expands the original testbed and incorporates substantial quantities of new technology components, while enabling the introduction of additional technologies in the future. Because of the need to continue to use the original testbed for evaluations already in progress while installing and configuring the new testbed components, the upgraded testbed was initially operated as two mostly independent systems. While nearly independent, these systems shared common storage resources and shared most of the same interconnect and SAN fabrics.

The updated testbed is configured as a Linux parallel scientific cluster, with a management node, a core set of 16 new dedicated compute nodes, a set of six special-purpose nodes, and a reserve of five auxiliary compute nodes from the original testbed. The Fibre Channel SAN fabric was expanded extensively with the addition of two 2 Gb/s FC switches. The Gigabit Ethernet used as message passing interconnect for parallel jobs was also expanded to support the increased number of nodes and iSCSI testing. A picture of the updated testbed appears as Figure 5. The updated configuration is shown in Figure 6.

The following major components were added to the testbed as part of its technology update:

- 22 new technology nodes: 16 in 2U cases and six in 4U cases (these are described in greater detail later in this section)
- two 16-port 2 Gb/s Fibre Channel switches
- one 16-port Extreme 5i Gigabit Ethernet switch
- one EMC Clarion CX600 disk subsystem
- one InfiniCon ISIS InfinIO 7000 fabric bridge
- one Cisco SN5428 iSCSI Storage Router fabric bridge
- one Myrinet 2000 eight-port switch and eight Myrinet 2000 Host interface cards
- five Intel PRO/1000 T IP Storage Adapter iSCSI HBA/NIC cards

- replacement Gigabit Ethernet cards for the original nodes
- replacement 2 Gb/s Qlogic QLA2310 FC HBAs for the original nodes
- additional Gigabit Ethernet cards and FC HBAs for the new nodes
- one additional Mega Frame rack mount cabinet.

Figure 5. The updated testbed with the NetStorage in front.

Various additional support components were obtained such as KVM (keyboard, video, mouse) switches, network power switches, an uninterruptible power source, additional Rocket Port serial concentrators, and various Ethernet and fiber optic cables.

The new technology nodes all utilize the same motherboard and are configured similarly. The only differences between them are the sizes of the cases they are installed in. Sixteen of the new technology nodes were put in 2U cases in order to save space, eliminating the need to buy more than one additional cabinet. Six of the new technology nodes were put in 4U cases to allow standard-height-profile PCI cards to be installed for the Myrinet 2000 Host interfaces, Intel PRO/1000 T IP iSCSI cards, and early 1x InfiniBand HCAs for the InfiniCon InfinIO 7000 fabric bridge. The need to have PCI-X slots on the motherboard to support the high performance FC, InfiniBand, and Gigabit Ethernet cards completely dictated the class of motherboards and processors that were acquired, as the only motherboards available with PCI-X buses were relatively high-end server motherboards.

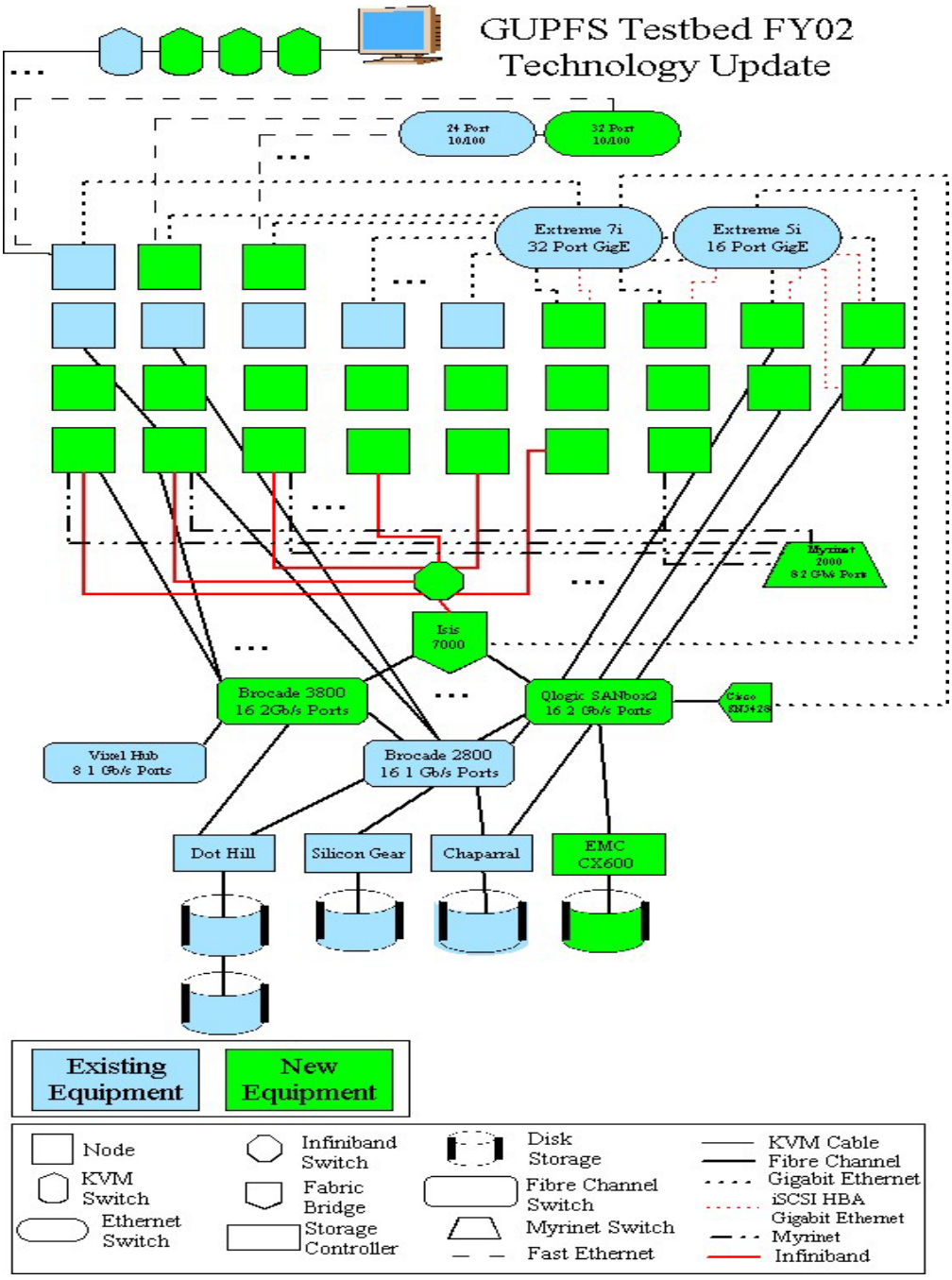

Figure 6. Updated GUPFS testbed configuration.

All nodes, regardless of the size of their case, are configured with:

- Supermicro P4DP6 motherboards with six PCI-X slots, two of which are 133 MHz capable
- dual 2.2 GHz Pentium IV Prestonia Xeon CPUs
- 2 GB of DDR PC2100 ECC memory
- dual on-board Intel PRO/100 Ethernet interfaces
- dual on-board U160 Adaptec SCSI controllers
- on-board VGA graphics

- one 36 GB Ultra 160 LVD 10K RPM SCSI disk drive
- one Qlogic qla2340 133 MHz PCI-X Fibre Channel HBA (low or standard profile)
- one Intel PRO/1000 XT 133 MHz PCI-X Gigabit Ethernet NIC (low or standard profile).

The identical base configuration of all the new nodes, including those intended to be special-purpose nodes, allows all of the new technology nodes to be used at times as compute nodes for the purpose of scalability testing. Four of the new nodes were configured to be special-purpose nodes. These special-purpose nodes have the same basic hardware and software configuration as the dedicated compute nodes. The four special-purpose nodes are configured to perform the following functions:

- code development and benchmark debugging
- metadata and lock manager services
- dedicated installation target for developing and testing new kickstart configurations
- storage server for testing distribution of shared file system with NFS gateways.

Two additional new nodes were reserved for transient special-purpose usage, such as running the InfiniBand subnet manager, which currently is supported only under Windows 2000.

The intent is for the original testbed nodes to take over many of the special-purpose node functions in the future, when the current series of evaluations being conducted on them are complete. This will free up more of the new technology nodes to operate as dedicated compute nodes.

In addition to the new technology testbed nodes, the testbed technology update included substantial additions to the testbed's SAN and message passing interconnect fabrics. These additions were partially driven by the need to scale these fabrics to accommodate the additional nodes, and partially driven by the evaluations and technology investigations the GUPFS project plans to conduct in the near term. The fabric components added include:

- 2 Gb/s Fibre Channel switches and node HBAs. Two 16-port 2 Gb/s FC switches were acquired: a Brocade Silkworm 3800 and a Qlogic SANbox2-16. Note that only fiber-optical cables can be used with the 2 Gb/s Fibre Channel technology; copper cables are not supported. The 2 Gb/s FC fabric will be integrated with the existing 1 Gb/s FC fabric. The EMC CX600 storage subsystem uses 2 Gb/s FC connections and will be connected to the new FC switches.

- The Gigabit Ethernet fabric was extended by the addition of a 16-port Extreme 5i Gigabit Ethernet switch. This new switch and the existing 32-port Extreme 7i switch will be trunked together over multiple interfaces. The resulting single Gigabit Ethernet fabric will be used as a message passing interconnect, for testing iSCSI storage transfers, and as a common bridge fabric between the Fibre Channel and InfiniBand fabrics.

- Two fabric bridge products were obtained to evaluate bridging between multiple fabric technologies. A Cisco SN5428 Storage Router provides a bridge between a Fibre Channel fabric and an Ethernet fabric. It allows iSCSI transfers between Ethernet attached hosts and FC attached storage. An InfiniCon InfinIO 7000 fabric bridge was also acquired. It allows

InfiniBand attached hosts to bridge from an InfiniBand fabric to both Ethernet and Fibre Channel fabrics.

Some additional special-use fabric components were included in the testbed technology update. These include:

- Five very early technology Intel PRO/1000 T IP iSCSI HBA/NIC cards that present a SCSI HBA to the host they are installed in. They convert native SCSI bus operations to iSCSI transfers over Ethernet at a hardware level. These cards will be used to explore the performance, utility, and desirability of hardware offload cards compared to a purely software iSCSI implementation. These cards may also be used to allow the hosts to present resources they have access to as iSCSI target devices.

- A small eight-element Myrinet 2000 fabric was added to the testbed, in the form of one eight-port switch and eight Rev C Myrinet host interface cards. These cards will facilitate testing of advanced features of the InfinARRAY file system. They will also allow the integration of portions of the GUPFS testbed into the LBNL Alvarez cluster's Myrinet fabric. This will allow the InfinARRAY and Lustre file systems to be studied on a much larger scale.

The increased scale, many advanced technology components, and flexible and expandable design of the updated GUPFS testbed will enable many interesting and important evaluations to be conducted over the next several years. These evaluations should lead to the selection of the best and most appropriate component technologies for the rollout of a high performance shared file system during the second phase of the GUPFS project.

4 Current Testing Methodologies

4.1 I/O Subsystem Introduction and Overview

Figure 7 depicts a conceptual view of the envisioned GUPFS environment. On a host, the environment is divided into user space and kernel space. Most of the computational programs run in user space, while the kernel provides resource scheduling, file access, and other important operating system functions. For most scientific computational jobs, from the viewpoint of the applications, every layer beneath the I/O library layer is part of the I/O subsystem. In order to obtain a complete view of the I/O performance, performance on every layer of the I/O subsystem needs to be understood.

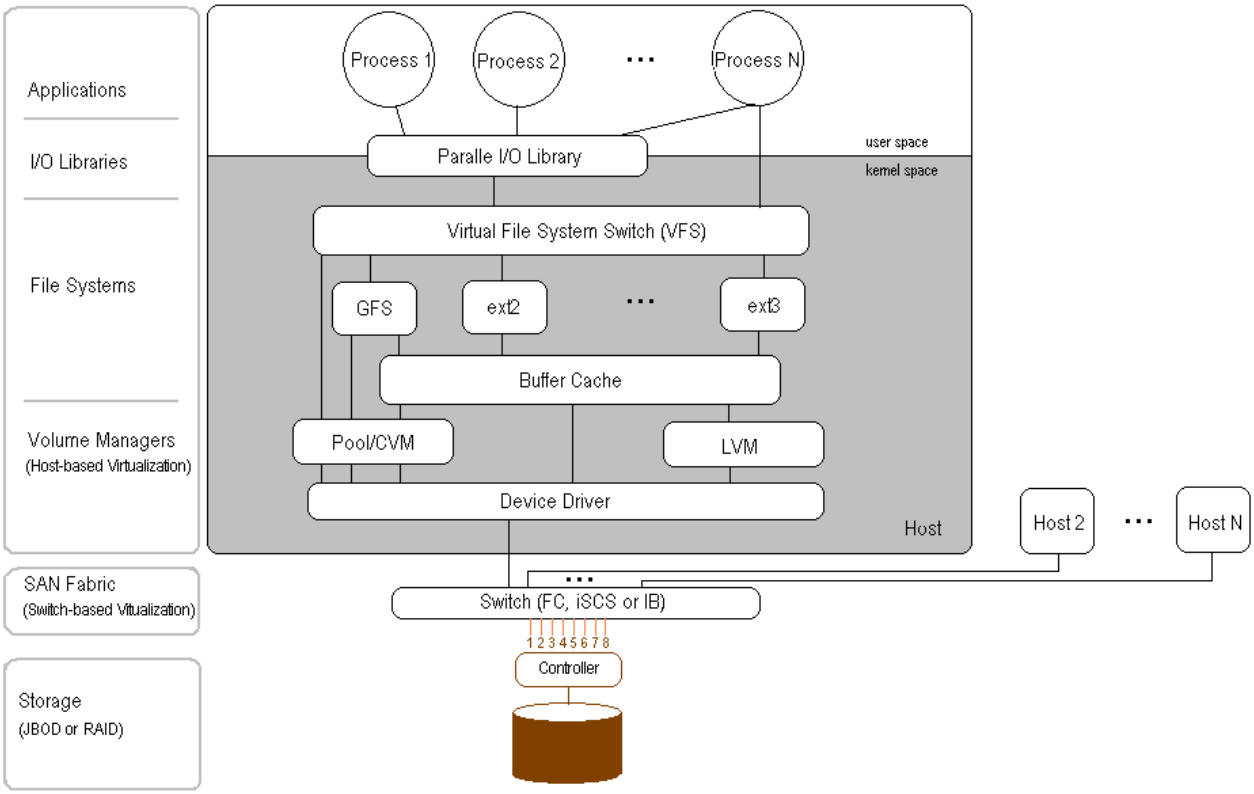

Figure 7 Conceptual view of the GUPFS environment.

When evaluating the parallel I/O performance, the layers of the I/O subsystem that need be studied include:

- Block I/O device (storage performance)
- Raw I/O to block device (including the HBA performance)
- Virtualization overhead
    - Host-based logical volume manager (Pool or cluster volume manager)

    - Switch-based virtualization (e.g., that provided by the Cisco SN5428 Fibre Channel to Ethernet bridge)
    - Storage controller and RAID
- I/O through a parallel file system
    - Scalability
    - Support of direct I/O
    - Caching issues
    - Locking issues
- Parallel I/O or application libraries
    - MPI-I/O
    - HDF5
    - NetCDF
- Application programs

When evaluating I/O performance, the two major measurements that are important to the GUPFS project are *aggregate performance* and *scalability,* although single-system performance is also of great interest. Aggregate performance measures the best possible performance each I/O layer can achieve, while scalability measures how each layer performs when the number of clients increases. The measurements can be done with different block sizes, different file sizes, different number of processes, etc.

To achieve a better performance or a higher capacity, storage (block-device) virtualization can also be used at some layers of the I/O subsystem. Storage virtualization can be implemented on the host computer (e.g., Logical Volume Manager (LVM) and Sistina's Pool utility), in a fabric switch, in a virtualization appliance (e.g., Chaparral's A8526 controller), or in a storage subsystem (e.g., DotHill SANnet, EMC CX 600, and Yotta Yotta GSX 2400).

Virtualization can also be implemented at the file system level by aggregating multiple file systems into one large virtual file system. A few examples of file system virtualization include HP's DiFFS, Maximum Throughput's InfinARRAY, Ibrix' SAN-based file system, etc.

Virtualization can add another dimension of complexity for the evaluation of I/O performance.

4.2 GUPFS Benchmarking Approach

File systems, and parallel file systems in particular, are extremely complicated systems and should be evaluated within the context of their intended use. In the commercial world, a file system may be evaluated by how it performs on a single or small number of critical applications. For example, a Web-serving content provider may not be interested in parallel write performance, since the primary role of the file system is to provide read-only access to data across a large number of servers without having to replicate the data to multiple farms.

By contrast, the NERSC HPC environment must support a large number of user-written applications with varying I/O and metadata performance requirements. Further, applications running today may not resemble the applications that will be running two years from now. Given

this diversity, the GUPFS project is taking a more general, multi-pronged approach to the evaluation of parallel file systems.

Initially, the GUPFS project will perform I/O scalability and metadata performance studies. Later, it will perform reliability and stress-test studies, and finally it will evaluate the performance with respect to specific I/O applications that emulate real NERSC user codes.

4.2.1 I/O Performance Studies

Evaluating the performance of a file system, and a parallel file system in particular, can only be done within the context of the underlying system and storage environment. If the underlying storage network or device can only perform at a rate of 30 MB/s, then we cannot expect sustained I/O performance through a file system to exceed this. In addition, for the case of a parallel file system, we need to understand the scalability of the underlying storage network before passing judgment on the file systems' ability to scale to a large number of clients. To aid in this portion of the study, we have developed a parallel I/O benchmark named 'MPTIO' which can perform a variety of I/O operations on files or devices in a flexible manner. Mptio is an MPI program in which each process spawns multiple threads to perform I/O on the underlying file or device. MPI is used to synchronize the I/O activity and to collect the per-process results for reporting. It does not use the MPI-I/O library. When acting through a file system, MPTIO can have all threads perform I/O to a single global file, or all the threads of a process can perform I/O to a single per-process file, or all the threads can perform I/O to distinct per-thread files. In addition, I/O can be performed directly to a block device or Linux raw device to help characterize the underlying storage devices and storage area network. Further, each thread can perform I/O on distinct, non-overlapping regions of the file or device, or multiple threads can perform overlapping I/O. The code can run five different I/O tests, including sequential read and write, random I/O, and read-modify-write tests. Aggregate and per-process results are reported. For a complete description of this code, see Appendix A.

We can baseline the performance of the storage network and a device as follows:

- First we measure the I/O rates from a single node to or from the device. We do this by writing a small file to the raw device such that the data will fit into the storage device controller's cache. Then we measure the rate at which we read it back. We repeat this test, varying the number of I/O threads in this single process job. At some point, the read rate will saturate, indicating we have hit a bottleneck either on the node, in the network, or on the device controller. If the bottleneck is on the node, there may be tuning parameters in the operating system that will improve the I/O rate; for example, eliminating the Linux bounce buffers by patching the kernel.

- Second, we scale up the number of processes (each on a separate node) and also vary the number of I/O threads per process. If the I/O rates do not improve past a single node, then the performance bottleneck is in some portion of the network or on the storage device. If they do improve, the first test gives us a good estimation of the peak sustainable I/O rate onto a single node. In this case, as the number of nodes increases, eventually the I/O rate will again saturate. This bottleneck is either in the network or on the controller.

- Third, if multiple paths exist through the network to the controller, we can use MPTIO to perform I/O through both paths. If the aggregate I/O rate saturates at the same level, then the bottleneck is in the controller and we know the maximal sustained I/O rate off the controller. If it saturates at a higher level, then the bottleneck was in the network, and adding multiple paths will improve performance.

Next we can measure the sequential I/O rates for data that does not fit into the controller cache. The controller will probably attempt to perform read-ahead and write-behind to hide disk latency; but as the number of I/O streams increases, the cache will start to thrash and performance will decrease. In addition, one can perform random I/O to get a handle on the worst-case performance of the device.

Logical volume managers can glue together multiple physical devices into one larger logical device and stripe I/O across the devices. Mptio can perform raw I/O on the logical device in a similar manner to baseline the characteristics of this device. If the logical volume manager is built into the file system and not directly available outside of it, MPTIO can emulate a poor-man's volume manager by performing I/O to multiple raw devices simultaneously.

Once a profile of the storage device and network is complete, a file system can be built over the device. We can then perform I/O and scalability studies over the file system. Since most file systems cache data in host memory, large I/O requests will have to be used to minimize the effects of caching. For example, if we observe better performance through the file system than to the raw device, we can conclude it is the effect of data caching on the node. In addition, one can compare the performance difference between multiple processes writing to the same file verses each writing to different files. If the performance difference is substantial, it will likely be due to write lock contention between the processes. This might be alleviated if the file system supports byte-range locking so that each process can write its own section without obtaining the single (global) file lock. If the file system supports Direct I/O, then one can compare I/O performance through the file system with what was measured to the raw device. The difference will indicate the amount of software and organizational overhead associated with the file system. For example, file block allocation will probably be distributed across the device, resulting in multiple non-contiguous I/O requests to the device. Such I/O performance can degrade as the file system fills and block allocation is more fragmented. File system build options and mount options may also play a substantial role in the performance.

4.2.2 Metadata Performance Studies

File systems are elaborate data structures that present a hierarchical directory structure, extensible files, and other useful objects to the user. For example, a file is represented by a data structure, often called an *inode,* that maintains information about the file, such as its size, who owns it, when it was last modified, and where the data in the file is located. A directory is a data structure that maintains an association of file names to inodes. These data structures are mapped onto the addressable blocks or objects presented by the underlying storage system. The file system must keep track of which underlying device blocks are in use and where the various portions of the file system data structure are stored on these device blocks or objects. Additional data structures are required to maintain this lower-level block allocation information. As operations are performed on the file system, such as creating a directory or file, or appending to a file, the data structures must be updated and maintained in a consistent manner. Creating a file would require allocating and

initializing an inode, modifying the directory data structure in which it will reside, and possibly allocating storage space for the contents of the file. All of these operations may require allocating or rearranging underlying storage units to hold the new or modified structures. These data structures are referred to as file system *metadata.*

In a typical (local) file system, the metadata is often heavily cached in the system memory and updated in an order such that, in the event of a system crash, the file system would be reconstructible from the on-disk image. Modern file systems maintain transaction logs or journals to keep track of which updates have been committed to stable storage and which have not. Each of the data structures generally contains one or more locks such that operating system actors wishing to modify the structure do so in an atomic manner, by first acquiring the lock, modifying the structure, and then releasing the lock. Some file system operations require modifying many of the structures and thus the actors must acquire multiple locks to complete the operation. In a parallel file system, multiple clients running on different machines under different instances of an operating system will want to modify these data structures (to perform operations) in an unpredictable order. In these cases, where the metadata is maintained and who controls it can have a dramatic effect on the performance of a file system operation.

There are currently two main approaches to managing metadata in parallel file systems: symmetric (distributed) and asymmetric. In a purely symmetric file system, all the file system clients hold and share metadata. Clients wanting to perform an operation must request locks from the clients holding them via some form of distributed lock manager. In an asymmetric file system, a central (dedicated) metadata server maintains all the information. The clients request updates or read and write access to files. Clearly, this latter case is easier to manage but establishes a single point of failure and may create a performance bottleneck as the number of clients increases. Note that in both cases, once a client is granted read or write access to a file, it accesses the file data directly through the SAN.

One aspect of evaluating a parallel file system is how well it performs various metadata operations. Clearly, where and how the metadata is maintained will have a substantial impact on the performance of various file system operations. Clients have to send messages to other nodes in the cluster requesting locks, or asking for operations to be performed. The latency of the interconnection network and the software overhead of processing the communication stack will be a major portion of these costs. Apart from the issues of parallel file systems, a portion of the metadata performance will have to do with how the data structures are organized internally. For example, some file systems will maintain a directory structure as linear linked lists whereas others will use more sophisticated schemes, such as hash tables and balanced trees. Linear lists are easy to implement, but access and update performance degrades rather seriously as the number of directory entries increases. The other schemes provide near-constant or log-time access and update rates and perform well as the directory grows. This, of course, is at the expense of a more complicated implementation. Another example is how the file system maintains information about which underlying blocks are free or in use. Some systems use a bitmap whereby the value of the bit indicates the availability of the block. Other systems use extent-based schemes whereby a small record can represent the availability of a large region of contiguous blocks.

In our study, we would like to measure how the various parallel file systems perform with respect to certain metadata intensive operations. To this end, we have begun development of METABENCH, a file system metadata benchmark code. This is a parallel application that uses MPI to coordinate processes across multiple file system clients. The processes perform a series of metadata operations, such as file creation, file stating and file utime and append operations. Details on the current state of METABENCH can be found in Appendix A. The tests are generally performed in a controlled manner, whereby one process creates the test environment and the other processes participate in the test. This prevents the process that created the environment from having an unfair advantage over other processes due to the fact that metadata information may be cached on that node. Tests are repeated both in a single directory and in separate per-process directories. The first case would likely have higher lock contention between the processes, and the timing differences would therefore be of interest. With this code, we hope to uncover and compare the metadata strengths and weaknesses of the various file systems under study.

4.2.3 Reliability Studies

Although the performance of a parallel file system is important, reliability is paramount. A file system that loses or corrupts user data is less than acceptable. We will need to develop a series of tests to evaluate how each prospective file system performs under heavy loads and failure conditions. This work has not yet begun.

4.2.4 Emulated User Applications

Although micro benchmarks such as MPTIO and METABENCH can provide a wealth of information about how a file system behaves under controlled conditions, the true test of a file system is how it performs in a real user environment. Testing in a real user environment is difficult because the NERSC user community is large, with a diverse collection of codes that evolve over time. In addition, the codes are complex and may not easily be ported to our test system, or even scale down to that size. Further, the I/O and file operation portion of the code may only consume a small portion of the run-time, so attempting to run the actual application on the testbed would be inefficient. In order to address these issues, we plan to develop a small collection of I/O applications that emulate the I/O and file management behavior of real NERSC user applications. This will be a time-consuming task and will only be successful with the aid of the user community. As mentioned in Section 1.3.2 of this report, we have begun an informal survey of some of the larger NERSC projects to understand their I/O requirements. As a part of this effort, we will select a few applications and solicit the users to help us create an I/O benchmark that emulates their code. We already have a commitment from Ricky Kendall at Ames Lab to produce a code that emulates the three I/O modes of NWChem.

One additional complexity when dealing with user applications is the use of I/O libraries. There are several I/O libraries in heavy use by certain segments of the NERSC user community. These include NetCDF, HDF5, and MPI-I/O. NetCDF is a serial I/O library in heavy use by the climate and atmospheric modeling community. It has a self-describing file format for ease in moving binary data between various architectures. The NERSC User Services Group is developing a parallel version of this library that used MPI-I/O. HDF5 is another library based on a self-describing file format, developed at the University of Illinois. It appears to have a more flexible and extensible format than NetCDF and is also built over MPI-I/O. MPI-I/O is a subset of the MPI2

specification in which file formats are abstracted in a manner similar to MPI user-defined data types.

One problem with using MPI-I/O benchmarks for a file system performance study is that the performance can be more dependent on the implementation of the library than on the file system. For example, the implementation may reorganize the data via message passing and perform I/O on only a single or small number of nodes. These organizational decisions should be made with regard to the characteristics of the underlying file system. An MPI-I/O application may perform well on an IBM SP where the implementation is tuned for working with the GPFS file system, but not perform well on a Linux cluster where the library implementation is generic and possibly not tuned for the file system under investigation. Such benchmarks are important to the user community, but the results must be interpreted carefully so as to understand the effects of each layer in the software stack.

4.3 Benchmarking Codes

The benchmarks used in the performance evaluation include many of the serial (e.g., LMbench, Bonnie, tiotest, etc) and parallel I/O benchmarks (e.g., pioraw, Flash, btio, etc.) that are available in the community. However, many of these benchmarks address only some aspects of the system performance. To augment the set of tools for the performance evaluation of the storage and file systems, the MPTIO and METABENCH benchmarks were developed in 2002 for the GUPFS evaluation. A complete description of these benchmark codes is provided in Appendix A.

5 Current Results Summary

5.1 Node and HBA Performance

In this section we present the Qlogic QLA2340 Fibre Channel HBA performance results based on the **`lmdd`** utility.

The tests were performed using an EMC CX 600 array connected to a single host node. The CX 600 has two storage controllers, each with 1,400 MB read cache and 2,000 MB write cache. Each controller has four 2 Gb/s Fibre Channel ports, but only one port was used in these tests. This port was connected to the host using the QLA2340 HBA. The host was a dual 2.2 GHz Pentium IV system with 2 GB memory running a Linux 2.4.8-10 RedHat kernel using a Qlogic driver with the bounce buffer bypass patch, which is discussed below. Figure 8 shows the single-port read and write performance with different I/O sizes.

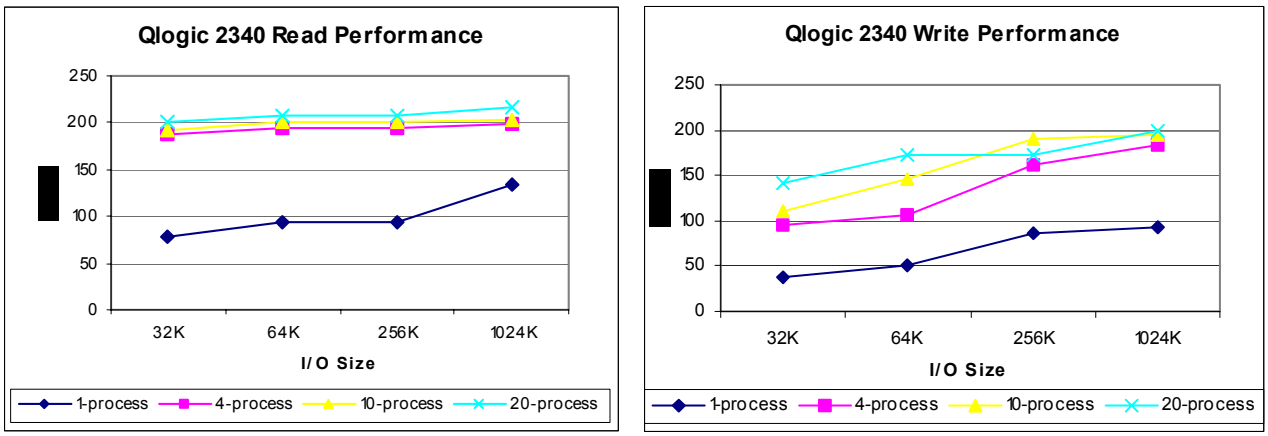

Figure 8. Qlogic 2340 read and write performance.

The read results seem to indicate that, except for the single-process read tests, the QLA2340 HBA was able to achieve a read performance close to 200 MB/s with multiple processes using an I/O size larger than 64 KB. The best performance was achieved with the 20-process test using an I/O size of 1,024 KB. The write results seem to indicate that the QLA2340 HBA was able to achieve a write performance that is close to 200 MB/s. The best write performance was also achieved with the 20-process test using an I/O size of 1,024 KB.

During our early tests, the best single-node performance we were able to achieve on a single HBA was about 70 MB/s. The HBA we were using at that time was a 1Gb Qlogic QLA2200 HBA, which was capable of sustaining an I/O rate greater than 100 MB/s. With the same setup, the same test could generate different performance results. We also observed the same poor performance when we were doing the beta evaluation of the Yotta Yotta GSX 2400 storage. Yotta Yotta reported that this could be caused by the known bounce buffer problem on a host that has larger than 1 GB system memory.

The memory that is above 1 GB is known as high memory. When DMA I/O is performed to or from this high memory, an area that is known as a *bounce buffer* is allocated in low memory. During data transfer between a device and high memory, data is first copied through the bounce buffer. It has been observed that systems with intense I/O activity and a large amount of memory can have a performance issue. There are two possible ways to avoid the bounce buffer problem: one is to reduce the size of system memory; the other is to use the latest kernel and the driver with the bounce buffer bypass patch.

The 2-4-18-x kernel that we were using during the I/O tests did include the bounce buffer patch. However, the Qlogic v6.1b2 driver that we were using did not have the bypass patch. As a result, even with the right kernel, we were still experiencing poor performance caused by the bounce buffer problem. To avoid the bounce buffer problem, the driver was modified to know about high memory[3]. Figure 9 shows the performance difference with and without the bounce buffer patch.

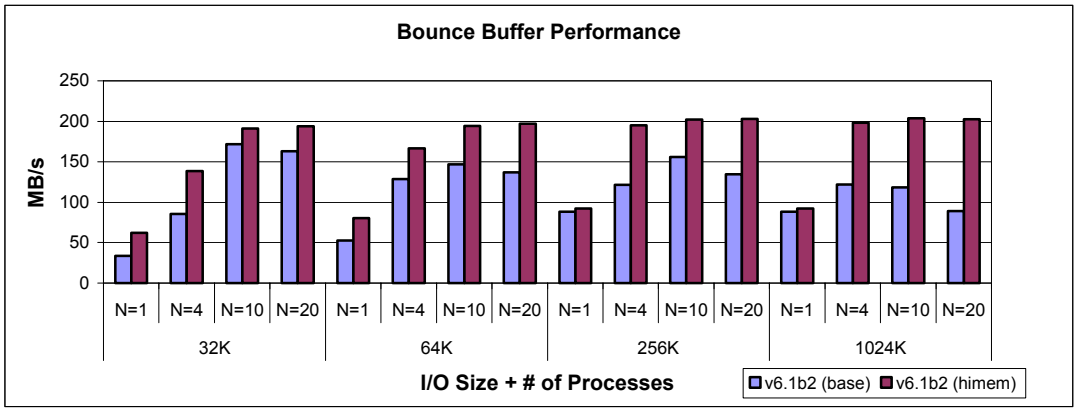

Figure 9. Bounce buffer performance.

The test was a simple N-process, sequential read test using a file size of 100 MB. The new driver made a significant performance difference, sometimes more than 100%. The new driver was later used in all the I/O tests when the QLA2340 HBA was used. The detailed results also show that the new driver had a better load distribution among multiple processes. With the base driver, some of the N processes often took longer than the rest of the processes to finish. With the new driver, all N processes finished at about the same time.

5.2 Block and Raw I/O Performance

In this section we profile the raw I/O rates to the Silicon Gear storage device. The tests were performed with a Silicon Gear RAID array using eight nodes on the GUPFS testbed. The Silicon Gear has two controllers, each with 256 MB of on-board cache memory. Each controller exports a RAID 5 (5+1) array with a stripe width of 320 KB. The controllers are connected to the Brocade 3800 FC switch via separate 1 Gb links. The client nodes are connected to the Brocade 3800 switch through 2 Gb links. Each client is a 2.2 GHz Pentium IV with 2 GB memory running a Linux

[3] The modification was a one-line change by Greg Butler. The new driver is labeled "v6.1b2 (himem)" in Figure 9.

2.4.18-10 RedHat kernel with a patch that eliminates the need for a kernel copy through a bounce buffer for device I/O.

We present both read and write performance for files that fit into the controller cache and for files that are larger than the cache. We also present results for performing I/O through one of the device controllers, and through both controllers simultaneously. We used Linux raw I/O that not only bypasses the system buffer cache but also uses the Linux kiobuf feature to bypass data copies into and out of the kernel. Each I/O test was repeated a number of times, varying the number of clients (processes) and varying the number of I/O streams (threads) involved in the test. In Figures 10 and 11, each curve shows the I/O scaling rate keeping the total file size and number of threads per client fixed, but varying the number of clients from one to eight. The X-axis represents the total number of threads (independent I/O streams) in the test and the Y-axis is the aggregate I/O rate. Note that in the two controller tests, we used the pool logical volume manager distributed with Sistina's GFS file system to stripe data across both LUNs, then built a Linux raw device over this logical volume.

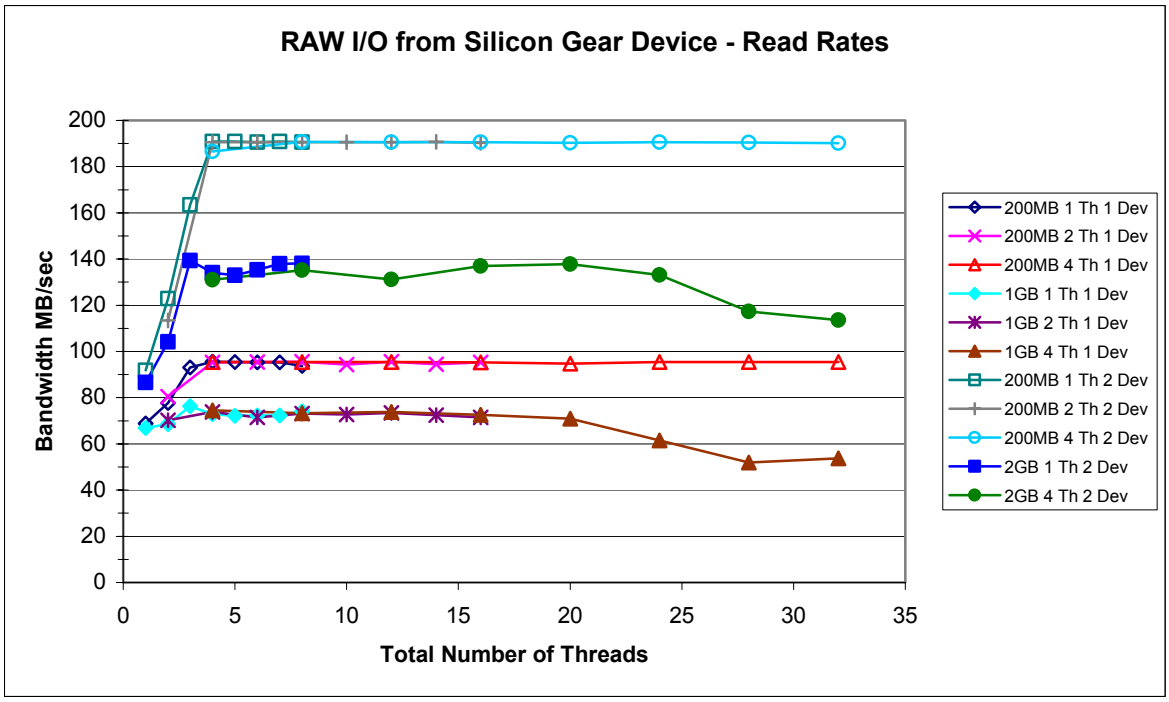

Figure 10. Raw I/O read performance.

In the first study, we look at read performance. Figure 10 clearly shows four families of curves: small files to a single controller, large files to a single controller, small files distributed between both controllers, and large files distributed between both controllers. The small file tests were conducted by populating the controller cache by writing the file just before the read tests were run. In these cases, the controller can service all requests without having to access the underlying disk array. As we can see, the maximum read rate from a single controller for data that resides in the cache is about 95 MB/s. It reaches this rate with four or more I/O streams regardless of the number of clients. This indicates that our client hosts are not limiting the read rate. Given that the controller has a 1 Gb/s Fibre Channel link, the maximum bandwidth through that link is about 125 MB/s. We

do not have multiple paths to a single controller, so we are not able to determine whether the network or the controller is limiting the rate. Note also that when small file reads are performed from both controllers, the rate saturates at 190 MB/s, double the single controller rate.
In similar tests, we observed that with six or more threads we can reach the saturation rate of 190 MB/s from a single client over a 2 Gb/s link.

When performing large reads, the I/O rate from a single controller stabilizes at about 74 MB/s then decreases to about 52 MB/s as the number of independent I/O streams increases above 20. In this case, the file cannot fit into the controller cache and must divide the cache between an increasing number of streams. As the controller attempts to perform read-ahead for each stream, eventually the cache begins to thrash and performance dips. We can see similar performance for large file reads from both controllers at rates slightly less than double the single controller rate.

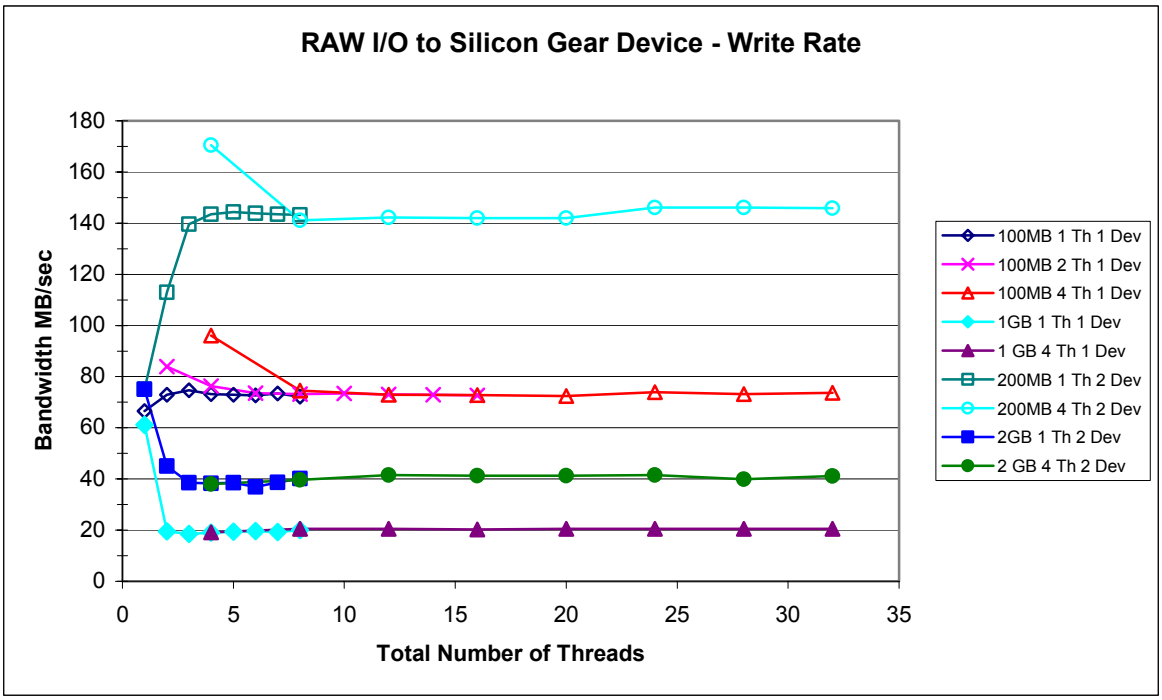

Figure 11. Raw I/O write performance.

Next we consider file write rates. When we began conducting these tests, we saw a large variation in the I/O rates between test runs. This was probably due to the fact that we had just populated the controller cache, then immediately ran another test. The controller was probably still busy issuing the write-back to disk from the first test. To mitigate this effect, we inserted a cache-populating read test from a high disk address between each write test. This had the effect of forcing the controllers to finish the disk write-back and read the requested data (which was ignored) before the following write test. As with the read test, we clearly see four families of curves in Figure 11, small and large writes to a single controller and to both controllers. We see that the small file writes saturate at about 72 MB/s to a single controller and double to about 142 MB/s to both controllers. For writes that do not fit into cache, the rates saturate at about 20 MB/s and 40 MB/s respectively. Note that small file writes have an odd behavior: we see better performance with multiple I/O

streams from a small number of nodes than with single stream (thread) performance from multiple nodes. This phenomenon only occurs when the number of client nodes is small. For large file writes, we see the performance quickly drop to the saturation rate as the number of I/O streams increases beyond two.

5.3 GFS 4.2 Performance

In this section we present some GFS 4.2 performance results. The tests were performed using a DotHill SANnet array from five of the older dual Pentium III nodes on the GUPFS testbed. The DotHill has two controllers, each with 1,024 MB of on-board cache memory. Each controller exports a RAID 5 (5+1) array with a stripe width of 320 KB. The controllers are connected to the Brocade 2800 FC switch via separate 1 Gb links. The client nodes are connected to the Brocade 2800 switch through 1Gb links (via 1 Gb QLA2200 HBA). Each of the client nodes was a 1 GHz Pentium III with 1 GB memory running a Linux 2.4.9 RedHat kernel with a GFS patch. The bounce buffer problem was observed constantly unless less memory was used.

The tests were done on the following six configurations for comparison:

- /dev/sdb — one of the two raw devices used in the pool definition
- pool (lo) — a pool device used for the gfs file system, the host had 512 MB memory
- pool (hi) — the same pool device, except the test was run on a host with 1GB memory
- gfs (dmep) — GFS 4.2 using hardware DEMP for locking; the host had 512 MB memory
- gfs (tcp) — GFS 4.2 with tcp lock daemon; the host had 512 MB memory
- gfs (tcp-hi) — GFS 4.2 with tcp lock daemon; the host had 1 GB memory

The results were obtained with the pioraw program running for 120 seconds using a file size of 256 MB. Figure 12 shows five-node performance results. The detailed performance results can be found in Appendix B.

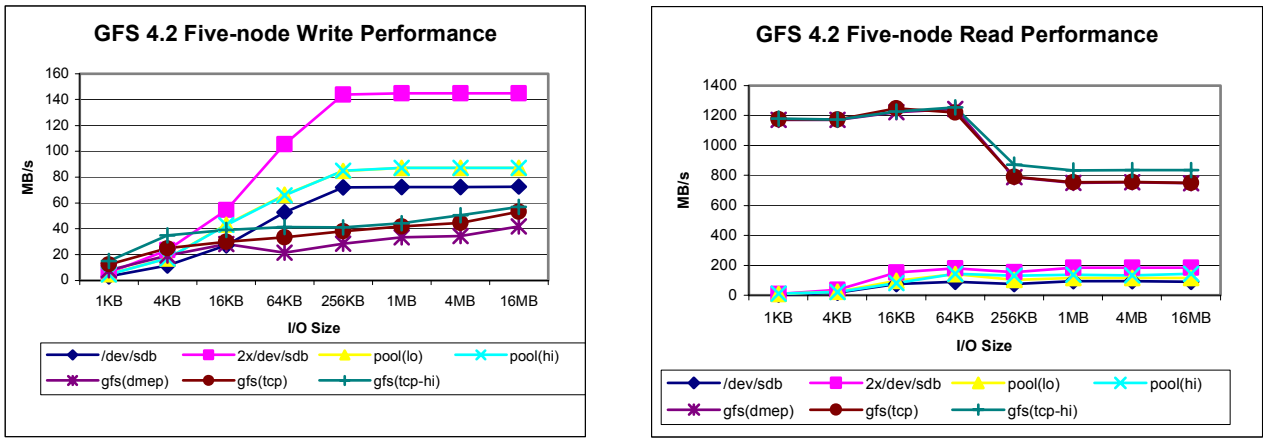

Figure 12. GFS 4.2 five-node performance results.

GFS 4.2 supports two lock managers: TCP-based and DMEP-based. The results seem to indicate that a GFS 4.2 file system using TCP lock manager outperforms, but only slightly, a GFS 4.2 file system using DMEP lock manager.

The pool device performance was far less than what the underlying devices could perform. It does not seem the bounce buffer problem was making any difference in the pool device performance. Figure 13 shows GFS 4.2 scalability for reading and writing 256 MB files.

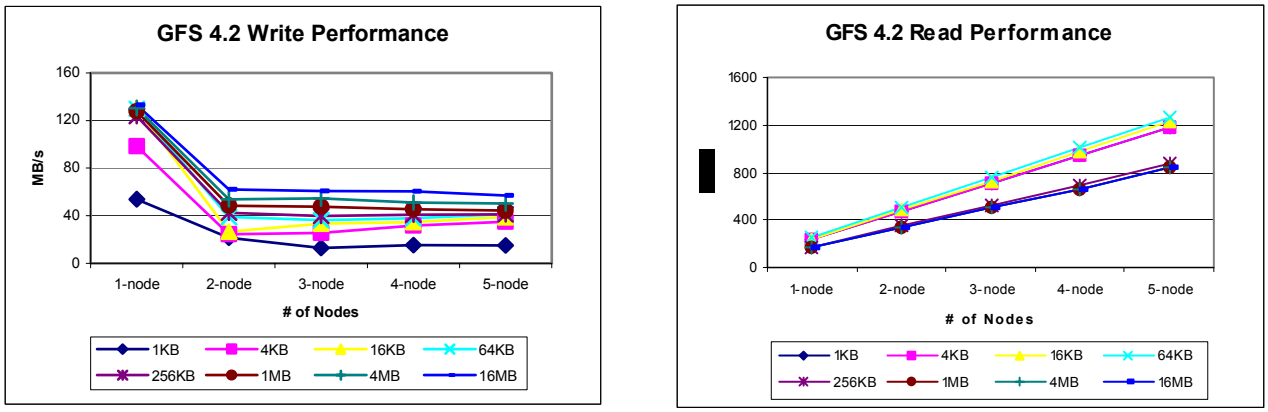

Figure 13. GFS 4.2 scalability results.

The results show a very poor write performance in a multi-node GFS 4.2 environment. With one node, GFS was capable of writing files at about 130 MB/s (with larger I/O size). However, when there were more nodes involved in the cluster, the write performance dropped to less than 70 MB/s and the rate remained at the same level independent of how many nodes were in the cluster. It may indicate that something inside GFS has serialized the write operations and was doing poorly in the multi-node GFS environment.

The 256 MB file size used in the pioraw tests was too small for the entire file to be cached on the test hosts. As a result, a very good and scalable read performance was observed. However, there was a drop in performance when the I/O size became greater than 64 KB. The reason for this drop is not clear without further investigation. Further study is also needed with a larger file size.

We were also interested in the pool device scalability. Figure 14 shows the scalability of the GFS 4.2 pool device. The results seem to show a very good pool device read and write scalability.

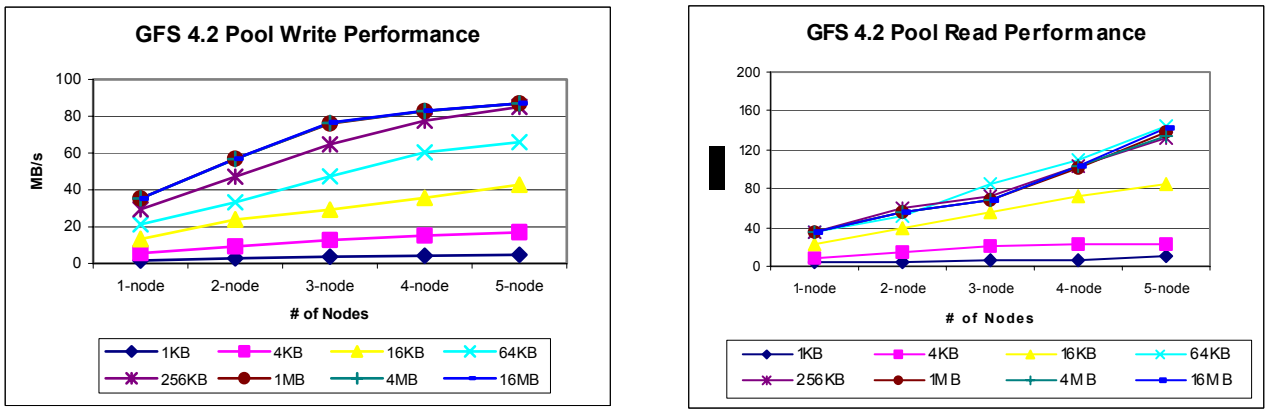

Figure 14. GFS 4.2 pool device scalability results.

5.4 GFS 5.1 Metadata Performance (Preliminary)

In this section we present some initial performance results based on the MPTIO and METABENCH benchmark. We must caution that these are preliminary results and have not undergone serious scrutiny.

The tests were performed using a Silicon Gear RAID array from eight of the dual Pentium IV nodes of the GUPFS testbed. The Silicon Gear has two controllers, each with 256 MB of on-board cache memory. Each controller exports a RAID 5 (5+1) array with a stripe width of 320 KB. The controllers are connected to the Brocade 3800 FC switch via separate 1 Gb links. The client nodes are connected to the Brocade 3800 switch through 2 Gb links (via 2 Gb QLA2340). Each client is a 2.2 GHz Pentium IV with 2 GB memory running a Linux 2.4.18-10 RedHat kernel with a patch that eliminates the need for a kernel copy through a bounce buffer for device I/O.

We performed a series of metadata performance studies on the GFS 5.1 file system using the METABENCH code. The tests were run on 1, 2, 3, 4, 6, and 8 client nodes to see the effects of scaling. In all of these cases except the first, `Process 0` was used only to prepare the test directories and files and collect results. That is, in the single client case, `Process 0` performed the setup work and the metadata tests. In the case where the number of processes is N > 1, `Process 0` just performs the setup and processes 1 … N-1 perform the metadata operations. It is instructive to compare the N = 1 and N = 2 cases. In both, only one process performs the metadata operations, but N = 1 often outperforms N = 2 because the directory and file setup was performed by the same process and the metadata may be locally cached.

Table 1 contains the aggregate file operation rates described in the following tests.

Table 1. Metadata Performance on the GFS 5.1 File System.

	N = 1	N = 2	N = 3	N = 4	N = 6	N = 8
Create-A	641	529	566	489	498	500
Create-B	614	548	988	1,295	1,362	1,562
Create-C	598	593	76	121	102	92
Stat-A	18,019	255	398	455	472	397
Stat-B	76,312	228	406	451	507	439
Utime-A	2,549	137	217	245	243	221
Utime-B	1,822	143	208	250	256	216
Append-A	1,232	143	214	274	263	227
Append-B	1,469	143	215	245	250	216
Untar	345	681	999	1,108	1,264	1,415

In test Create-A, we measure the zero-length file-creation rate in a single directory. The process with the highest MPI rank is selected to create 20,000 zero-length files in a directory created by `Process 0`. Timings are taken at 1,000-file intervals to see if the creation rates change as the directory grows. The aggregate creation rate varied between 489 and 641 files per second and did not change substantially as the number of files in the directory increased. The overall variation in the file creation rate between the test runs was about 10–20 percent.

In test Create-B, we measured the zero-length file-creation rate in which all the participating processes created their fair share of the 20,000 files in a separate, distinct directory. That is, when five processes participated, they each created 4,000 files in different directories. The values shown are the aggregate creation rates computed by dividing the total number of files created by the elapsed time for all processes to complete their work. Note that for N > 1, the creation rate is increasing but the rate is diminishing as the number of processes increases. Although the files are created in different directories, so the processes may not be contending for directory locks, they are probably contending for other resources, such as inode allocation. File creation on the same node where the directory was created (case N = 1) has a higher rate than if the files are created on a different node from the directory (case N = 2).

In test Create-C, we measure the time it takes for 20,000 files to be created in a single directory by all the participating processes. When N > 2, the performance is substantially lower than in the Create-B test because each process must contend for a single directory lock in order to create its files. Note that in the N < 3 cases, there is no directory lock contention between the processes, so the performance is substantially higher than for N ≥ 3.

In test Stat-A, we measure the time it takes to issue the UNIX stat system call on 20,000 files, which have been created and distributed into N directories by **`Process 0`**. Each participating process is assigned a directory and it must issue stat system calls for its share of the 20,000 files. In test Stat-B, all the files are created in a single directory, and each participating process is assigned an equal subset of the files to issue stat system calls on. The aggregate file stat rates between the two sets of tests do not vary in any substantial manner. This is because issuing a stat system call on a file requires reading but not modifying the metadata, and so there is less (if any) lock contention. The N = 1 case has a performance rate two orders of magnitude greater than the other cases. This is probably because this process recently created the files and the metadata is still cached in local memory. The major cost for both tests is probably the network latency of requesting the locks or metadata from **`Process 0`**.

The tests Utime-A and Utime-B are similar to the stat tests, except that the file access and modification times have been updated. The aggregate performance is about half that of the stat tests. This is probably because an exclusive write lock must be obtained and the metadata must be written (triggering, at minimum, a journal transaction) for the operation to complete. Again, the largest cost is probably network latency.

The Append-A and Append-B tests are also similar to the stat tests, except that each file is opened in append mode and 16 KB of data is written to each file. It is somewhat surprising that the append rates do not differ substantially from the Utime rates. Appending requires allocating data space for the file, updating the inode, and writing the data blocks. We see the relative costs between append and utime in the N = 1 case, where all metadata is local. The performance for the N > 1 case is likely dominated by the network latency and the cost of obtaining an exclusive write lock on the files.

Finally, in the Untar test, each process untars a 75.8 MB tar file containing 459 subdirectories and 9,188 files into separate directories. The tar file is placed in a local directory (/tmp) on each node

prior to the test. Process 0 creates the N working directories, then each of the participating processes enters their assigned directory and unpacks the archive. The values reported in the table are the aggregate file unpack rate. As opposed to the previous tests, where the amount of work was fixed and divided among the N processes, here the amount of work scales with the number of processes. This test more accurately emulates the kind of work individual users would perform in their home directories. The rate increases linearly at first, and then begins to diminish as the volume of metadata and I/O operations increases.

6 Vendor Contacts and Relationships

An important part of the GUPFS project is to develop contacts with component technology vendors and to develop relationships with those vendors having interesting, relevant products or product plans. The importance of developing such contacts and relationships lies in being able to influence vendor product directions to match the needs of the GUPFS project and scientific HPC. The GUPFS project has been very successful in developing vendor contacts and has developed relationships with a number of vendors.

One byproduct of developing contacts and relationships with vendors is receiving early pre-production (usually beta) equipment for evaluation and benchmarking. A specific advantage of conducting tests and benchmarks of such equipment is that it allows the GUPFS project to influence product design and features, to influence the future development of the product, and to influence the timetable for product features. These are all more difficult to achieve once a product is generally available.

6.1 Beta Tests Conducted

The GUPFS project was involved in three beta product tests during FY 2002. These beta tests were for the Dot Hill AXIS IP storage server, the Yotta Yotta NetStorager scalable storage product, and the InfiniCon ISIS InfinIO 7000 InfiniBand fabric bridge.

The Dot Hill AXIS was the first beta test conducted by the GUPFS project. The AXIS was received in December 2001. Although the AXIS product remained in the GUPFS testbed system for nearly four months, time pressures due other project activities limited the amount of testing that could be conducted.

The beta testing of the Yotta Yotta NetStorager [16] began in May 2002. This beta test, which is continuing into Q1 of FY 2003, has been very successful with regard to the GUPFS project being able to influence the functionality of the product and influence its development in directions beneficial to scientific HPC. The NetStorager has improved substantially, and is now delivering very good, scalable performance. The GUPFS project was able to apply the evaluation and benchmarking methodology that the project was developing to good effect during this testing.

Beta testing of the InfiniCon ISIS InfinIO 7000 [17] began in September 2002. Beta testing is expected to continue through Q1 of FY 2003, and it is very likely that it will continue as a long-term relationship with InfiniCon. The InfinIO 7000 provides a fabric bridge between InfiniBand and both Gigabit Ethernet and Fibre Channel. It utilizes InfiniBand HCAs on client compute nodes to communicate with both IP networks and Fibre Channel attached storage. The InfinIO will allow the GUPFS project to begin assessing the feasibility of InfiniBand as a SAN fabric and to begin exploring the bridging of multiple SAN fabric technologies together into a single SAN.

6.1.1 DotHill SANnet AXIS Evaluation

The DotHill SANnet AXIS box is a storage virtualization product with the capability of wide-area distribution via a proprietary iSCSI interface. It provides a number of services, including volume snapshots and replication for use in a distributed corporate environment. Our interest in the AXIS box is its capability to export the storage as a virtual SCSI device to remote clients. In particular, we were interested in whether we might be able to use this device to include a remote client as a node in a cluster file systems, such as GFS. We were also interested in the performance and reliability of system over an unpredictable and possibly high-loss wide-area interconnect.

The AXIS box we evaluated was a rack-mounted Pentium III with a Fibre Channel connection to the SAN and Gigabit Ethernet to the clients. The AXIS box scans the SAN and presents a list of all physical resources it discovers. Before exporting storage to a remote client, the administrator must construct logical resources from the underlying physical resources. The logical resource can be either a direct-mapped resource, in which the underlying storage is presented as is, or a virtual resource, in which physical volumes can be split or combined into large logical volumes. Volume management software allows the virtual volumes to be expanded or contracted as appropriate. The intent of the direct-mapped resource is to allow access to legacy storage.

Once constructed, logical resources can be assigned to remote clients. The client must be running an OS with a modified kernel in order to connect to the AXIS server and interpret the iSCSI commands. With Linux, this software consisted of several kernel modules pre-compiled for Linux 2.4.7, and several user space daemons. Once authenticated with the server, the client would be presented with what looked like a local SCSI device. The communication between client and server was performed by a proprietary reliable User Datagram Protocol (UDP).

In our testing, we built a Linux EXT2 file system on the exported SCSI device and ran several standard I/O benchmarks on the file system. Our client was a Pentium IV node with 512 MB of memory. The server was located in the GUPFS testbed at the Oakland Scientific Facility (OSF). We connected to the server over several different networks during the course of our evaluation. Our first test was over a 10 Mb/s half-duplex link at LBNL. This provided low performance but demonstrated that the AXIS box software was able to handle slow- and high-loss network. At one point, the local Ethernet error rate reached 90% and the system continued to function as advertised. We then moved the client to a full 100 Mb/s link at LBNL, then to a 1,000 Mb/s link at OSF, and finally as a node directly attached to the GUPFS Gigabit Ethernet switch using jumbo frames. The I/O performance scaled with the network speeds and, with jumbo frames, equaled that of a node directly attached to the underlying storage through a Fibre Channel link.

We were never able to test the client as a node in a GFS file system. At the time of our evaluation, the DotHill client required a Linux 2.4.7 kernel, and GFS 4.2 required a modified 2.4.9 kernel. In addition, all GFS clients must see the same SCSI devices, over which storage pools are constructed for both GFS data and configuration blocks. But we did not confirm whether this could be done for the remote clients using the AXIS direct-mapped logical volumes. The AXIS server interpreted the SCSI commands from the clients before issuing the requests to the underlying storage devices. At the time of our testing, the AXIS software did not implement the SCSI DMEP extensions, so we would not be able to use the hardware DMEP locking protocol in GFS. We could, however use the IP lock manager. The biggest issues with using the IP lock manager were secure network access

and subnet management. GFS and other cluster file systems assume the cluster is centrally located and has a local subnet isolated from the outside world. This subnet is used for GFS management, lock traffic, and STOMITH (or fencing) methods used to exclude a failed client from rejoining the cluster with stale lock data. The only reasonable way to create this subnet would have been to set up a virtual private network between the server site's internal management network and the remote site. In the end, we had to return the unit to DotHill before any of these issues were resolved.

6.1.2 Yotta Yotta NetStorager GSX 2400 Beta Evaluation

A major objective for the GUPFS project's FY 2003 plan is to demonstrate 1 GB/s sustained file system performance. In order for a file system to attain the 1 GB/s sustained performance goal, the underlying storage must be capable of sustaining more than 1 GB/s. For a shared-disk file system, all hosts have to connect and access the same underlying device simultaneously. This 1 GB/s goal has been difficult to achieve when the I/Os are performed from multiple host initiators. A major objective of the Yotta Yotta evaluation is to study whether GSX 2400 was able to break the 1 GB/s barrier and how GSX 2400 scales when the number of clients increases.

The tests were performed using a Yotta Yotta GSX 2400 array from seven of the GUPFS testbed's dual Pentium IV nodes. The GSX 2400 has four controller blades, each with 4096 MB of on-board cache memory. Each controller blade has two 2 Gb/s ports. The ports are connected to the Qlogic SANbox2 switch via separate 2 Gb/s links. The client nodes are connected to the same SANbox2 switch through 2 Gb/s links (via 2 Gb/s QLA2340 HBA). Each client is a 2.2 GHz Pentium IV with 2 GB of memory, running a Linux 2.4.8-10 RedHat kernel using a qlogic driver with the bounce buffer bypass patch. Figure 15 shows the single-port read and write performance with an I/O size of 1,024 KB using the `lmdd` utility.

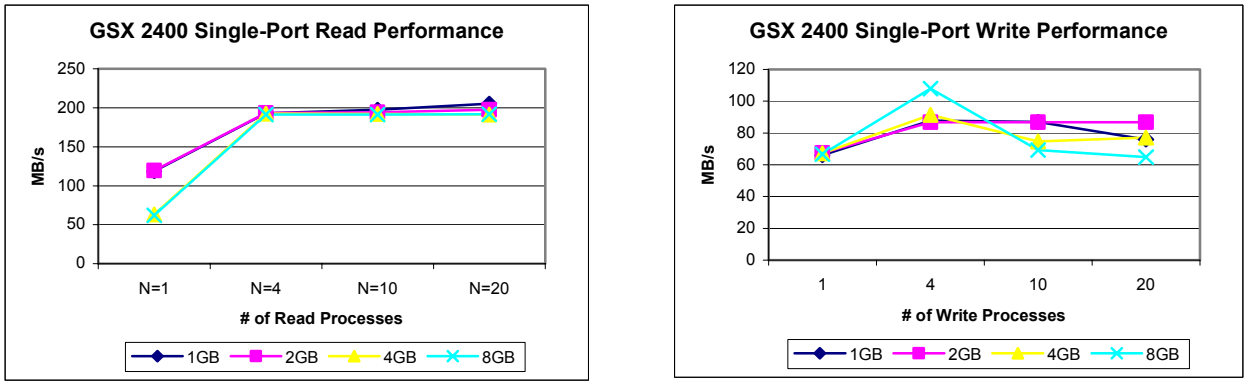

Figure 15. Yotta Yotta GSX 2400 single-port read and write performance.

Each I/O test was run twice to study how caching, read-ahead or write-behind implementation on GSX 2400 may affect the I/O performance. The cache on each blade is 4 GB and about 3 GB is usable for caching the user data. Therefore, 1 GB and 2 GB files would be cached completely in the blade cache. Only the results from the second tests are shown in Figure 15.

The result shows that the best possible read performance on a single port of the GSX 2400 is about 200+ MB/s. It begins to saturate the storage port with four processes. The 200 MB/s remains

constant even for the larger file size. This seems to indicate that the GSX 2400 was doing read-ahead reasonably well to feed the data to the read processes.

For single-stream writes, the best performance, independent of the file size, is about 67 MB/s. The write performance is around 70–90 MB/s, except for the case of writing an 8 GB file with four processes. From the detailed results shown in Appendix B, the four-process write tests seemed to outperform other multistream write tests using 10 or 20 processes, independent of the file size or block size used in the tests. This degradation with 10 or 20 processes could be caused by the resource (e.g., memory) contention or scheduling overhead on the host side. However, this does not seem to be the case with 20 read processes. Further investigation is needed to understand this behavior.

The GSX 2400 test unit has four blades with two ports on each blade. However, due to the stability of the test environment, we were only able to set up seven hosts connected to seven ports on the GSX 2400. Figure 16 shows the GSX 2400 scalability while reading 128 MB and 1 GB files, with one and four processes.

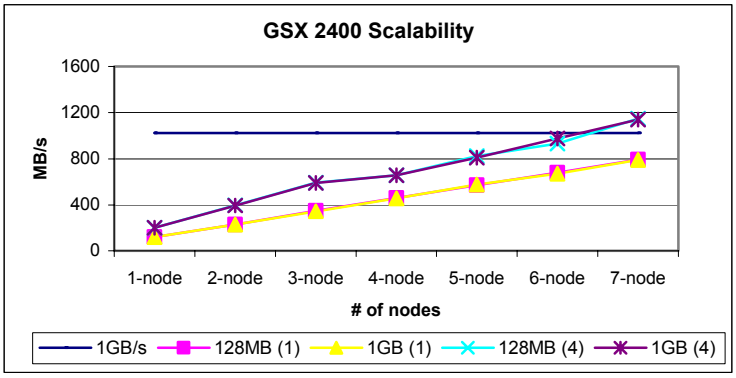

Figure 16. Yotta Yotta GSX 2400 scalability.

The performance results show very good scalability when the number of clients increases. With seven host connections, and with four read processes on each host, we were able to achieve the 1 GB/s objective.

6.2 Vendor Contacts

The GUPFS project personnel had extensive contacts with vendors during FY 2002. These contacts included vendor presentations, both onsite and at vendor facilities, discussions with vendors at conferences and expositions, and telephone calls. Some contacts did not result in anything useful or in follow-up contacts. Other contacts led to extensive and ongoing relationships, beta testing of equipment, and potential collaborations.

The vendors with whom the GUPFS project had contacts in FY 2002 are listed below, organized by the technology arena that applies to them. Their products are noted briefly, and if any relationship developed, it is noted.

6.2.1 File System Technology Vendors

- Maximum Throughput — InfinARRAY file system, NAS storage systems
- PolyServe — PolyServe cluster file system
- Cluster File System Inc. (Peter Braam) — Lustre file system
- ASCI PathForward — Lustre (SGSFS) file system
- Sistina — GFS shared-disk file system
- MacroImpact — SANique shared-disk file system
- Ibrix — a federated file system
- Exstore — a shared file system

6.2.2 Storage Technology Vendors

- Agile Storage — NAS storage, virtual NFS file systems
- Blue Arc — NAS, NFS acceleration in silicon, have contacts with Peter Braam and involved in the Luster OST arena, may develop into a useful contact, seem interested in a relationship.
- Dot Hill — High reliability storage subsystems, vendor for DMEP-capable controller purchased for original testbed, provided AXIS iSCSI server for beta evaluation; relationship has grown stale.
- EMC — Enterprise class storage
- Maximum Throughput — NAS storage systems
- Storage Computer — High performance SAN storage server capable of connecting to several different fabric technologies, potentially most useful as a fabric bridge; several meetings over the year.
- XIOtech
- Yotta Yotta — scalable storage system
- 3Par
- Zambeel — NAS storage, integrated NFS file systems

6.2.3 Fabric Technology Vendors

- Cisco — Ethernet, iSCSI, and Fibre Channel vendor, will to work with NERSC to do product evaluations, may participate in FC/iSCSI beta evaluation at end of CY 2002, potential for good relationship.
- InfiniCon — InfiniBand to Gigabit Ethernet and Fibre Channel bridge
- Mellanox — InfiniBand component vendor, producer of early reference host channel adapters (HCAs) and switches, multiple meetings, on their radar.

6.2.4 Other Vendors and Contacts

- Angstrom
- Cray Inc.
- IBM
- Unlimited Scale

7 Future GUPFS Project Activities

This section describes future planned activities, including tracking new and alternative architectures; near-term evaluations of file systems, storage technologies, and SAN fabric technologies; and longer-term investigations and activities.

7.1 Tracking New and Alternative Architectures

The GUPFS project has been focusing on "traditional" SAN-based shared-disk file systems. Such shared-disk file systems, of both symmetric and asymmetric architecture, have been the most common SAN-based file system technology. SAN-based shared-disk file systems are attractive because they do not share the inherent architectural limitations of traditional client-server network-attached storage (NAS) file systems, such as NFS and CIFS (Common Internet File System).

With the recently increasing commercial interest in large-scale shared file systems, several promising alternative shared file system architectures have appeared. These are the Lustre file system, the InfinARRAY file system, and the Ibrix file system. All three of these shared file systems perform storage transfers between client systems and storage servers over IP networks. Other than utilizing IP networks, these novel file systems have nothing in common with traditional NAS file systems. Each of these new shared file systems is targeting high aggregate performance in a cluster environment with a large number of clients. All of them are currently still under development, but early versions are becoming available for testing. The GUPFS project plans to track their development carefully and conduct periodic evaluations to determine their suitability for deployment at NERSC, and to determine whether they are better solutions than SAN-based shared-disk file systems.

7.1.1 Lustre File System

Lustre is unique in being an object-oriented distributed file system that is designed to scale to support tens of thousands of clients [18,19,20,21]. It is being developed as open source software under a contract awarded to HP and Cluster File Systems, Inc. by the ASCI PathForward SGSFS program. Intensive development of Lustre is under way and is scheduled to continue through CY 2005.

Lustre utilizes a metadata server (MDS) to manage file system namespace operations. It uses relatively lightweight software on each client system to communicate with the MDS and to access data over standard networks and interconnects. Data storage management and file system I/O transfers are done utilizing object storage targets (OSTs), which serve attached storage to Lustre client systems. OSTs also handle file locking for the objects they contain. Lustre is designed to allow thousands of OSTs to be used to achieve hundreds of gigabyte per second sustained aggregate I/O bandwidth. Currently OSTs are Linux-based systems, but third-party implementations are on the horizon. Lustre uses the Portals open networking protocol, originally developed at Sandia National Laboratory, to perform high performance network data transfers. The Portals network stack runs on clients on top of a network abstraction layer (NAL) that allows Lustre to be network-neutral and support multiple network technologies. Portals allows Remote Direct Memory Access (RDMA) and OS-bypass to be used with data transfers for increased performance.

The Lustre roadmap shows support for Myrinet, Fibre Channel, and InfiniBand being added over the next several years. Initially, Lustre will support TCP networks and Quadrics interconnects. An initial, very limited but functional release of Lustre is planned for Q1 FY 2003. More functionality will be rolled out as development continues over the next four years. Currently only Linux implementations of Lustre clients and servers are shown on the roadmap.

The ASCI PathForward SGSFS project under which Lustre is being developed shares some similar scalability and performance goals with the GUPFS project. The GUPFS project will track the progress of Lustre carefully as it is developed, and will periodically conduct evaluations of it to assess its suitability for deployment at NERSC.

7.1.2 Maximum Throughput InfinARRAY File System

Maximum Throughput's InfinARRAY file system also uses commodity IP networks to perform I/O transfers. The InfinARRAY file system is a hybrid file system. It is based on federating the individual file systems owned by storage processor (SP) systems and presenting this federation as a single file system to client systems over IP networks through Virtual Filesystem Switch (VFS) layer software running on the clients.

Maximum Throughput claims that InfinARRAY federates the underlying local file systems of the SPs in such a fashion that metadata lock management traffic and data synchronization issues are nearly eliminated. Although InfinARRAY uses IP networks for data transfers and metadata operations, it is claimed that through the use of multiple SPs and federating their file systems it does not suffer from the architectural and performance limitations of NFS.

The SPs are standard Intel Linux systems running special software. These systems serve their local file systems, which may be on either locally attached disk storage or SAN-attached disk storage. The SPs do not share their storage with each other.

Development of the InfinARRAY file system is in progress, and initial versions should be available for evaluation in early FY 2003. The GUPFS project is planning to conduct an evaluation of the InfinARRAY file system to determine its suitability and performance.

7.1.3 Ibrix File System

The Ibrix file system is also a federated file system distributed over IP networks. It federates individual file systems belonging to storage engines (SEs). It differs in implementation details from the InfinARRAY file system, but has many similarities. Ibrix achieves performance and scalability by aggregating SEs, which are back-ended by SAN-based storage. Unlike InfinARRAY SPs, Ibrix SEs will utilize specialized hardware.

The Ibrix file system is also currently under development. Initial versions should become available in FY 2003. When they do, the GUPFS project is interested in conducting an evaluation of Ibrix.

7.2 Near-Term Investigations

With the development of an evaluation methodology and benchmark suites, and with the updating of the GUPFS testbed system, the GUPFS project now has the necessary tools in place and is ready to proceed with a substantial number of investigations and evaluations during FY 2003. Investigations and evaluations are planned in all three critical component technology areas: shared file systems, storage, and SAN fabrics.

7.2.1 File Systems

The major focus of the GUPFS project in the near term (FY 2003) is to investigate and evaluate a large number of shared file systems. The GFS file system was evaluated in FY 2002. As new versions of GFS become available, they will be evaluated to track GFS's progress. In general, the GUPFS project plans to conduct periodic evaluations of each of the shared file systems that show promise in order to track their continued progress, and to give feedback to the vendors about the performance of new versions and additional features that are required.

During FY 2003, the GUPFS project currently plans to conduct evaluations of the following shared file systems:

- the ADIC StorNext (CVFS) file system
- the Lustre file system
- the Maximum Throughput InfinARRAY file system
- the IBM GPFS file system on the SP and Linux.

This list of file systems to be evaluated is not exhaustive. There are a number of other file systems that the GUPFS project is interested in evaluating within the next year or two. These include the MacroImpact SANique file system, the Ibrix file system, the PolyServe file system, and the SANergy LAN-based file system. These and other file systems will be evaluated based on their availability and adequate time resources.

ADIC StorNext File System Evaluation

The ADIC StorNext file system, previously known as the CentraVision file system (CVFS), will probably be the first file system to be evaluated in FY 2003. An evaluation agreement with ADIC is in the final stages of being completed. ADIC has already installed the StorNext version 2.02 system on the GUPFS testbed and provided some onsite training. Once the evaluation agreement is finalized, evaluation can begin quickly. ADIC planned to release a new version of StorNext in December 2002 and is planning to have the GUPFS project evaluate this version also. Indeed, ADIC is very interested in establishing a long-term relationship with the GUPFS project and NERSC, and is very interested in working with NERSC to develop their product.

StorNext is unusual in the shared file system arena in that it already provides support for multiple operating systems — Linux, SGI IRIX, Microsoft Windows, and Solaris. The December release of version 3.0 was scheduled to add support for AIX 5.3. The multiple platform support provided by StorNext will allow the GUPFS project to test a shared file system in a truly heterogeneous hardware platform and OS environment, potentially involving Sun Solaris SPARC systems, an IBM AIX system in the form of the dev2 system, and an SGI IRIX system in the form of the NERSC visualization system.

The planned StorNext evaluations in FY 2003 are:

1. Evaluation of StorNext v2.0.2 in Q1, using only Linux clients in the GUPFS testbed.
2. Evaluation of StorNext v3.0 in Q2, using only Linux clients in the GUPFS testbed.

Based on the availability of hardware resources within NERSC and on available time, the GUPFS project would also like to conduct the following optional evaluations with StorNext:

1. Evaluation of StorNext v3.0 in Q2 or Q3 with GUPFS testbed Linux clients and Sun SPARC Solaris client(s).
2. Evaluation of StorNext v3.0 in Q2 or Q3 with GUPFS testbed Linux clients and an SGI IRIX client (visualization server).
3. Evaluation of StorNext v3.0 in Q2 or Q3 with GUPFS testbed Linux clients and IBM AIX 5.3 client(s) (possibly using the dev2 system).
4. Evaluation of StorNext v3.0 in Q2 or Q3 simultaneously with clients used in the optional evaluations 1–3 outlined above.

Lustre File System Evaluation

With the award of the ASCI PathForward SGSFS file system development contract to HP and Cluster File System Inc., there has been rapid progress on the Lustre file system. As of mid-Q1 of FY 2003, versions of Lustre are appearing that are stable and functional enough to begin testing. Lustre is likely to be the second shared file system to be evaluated in FY 2003 and frequent re-evaluations are expected as new versions come out. The updated testbed now has sufficient nodes to be able to use some of them as object storage targets (OST) for testing Lustre while retaining enough nodes as client systems for testing.

The planned Lustre file system activities and evaluations for FY 2003 are:

1. Initial build and installation of Lustre client, metadata manager, and OST software on the GUPFS testbed in late Q1.
2. Lustre software refresh and initial evaluation using only testbed resources in early Q2.
3. Lustre software refresh and repeat of evaluation to gauge forward progress in late Q3 or early Q4.

Based on the experience gained with Lustre during the initial evaluation, progress in Lustre development, and the availability of resources and time, the following optional evaluation activities will be attempted in FY 2003:

1. Evaluation of Lustre scaling properties with a large number of nodes, using the Alvarez cluster compute nodes as clients, and using GUPFS testbed nodes as OSTs with Myrinet interconnect between the OSTs and LBNL's Alvarez cluster.
2. Evaluation of Lustre scaling and performance properties using Alvarez compute nodes as Lustre clients and GUPFS testbed nodes as OSTs. This evaluation would be done using Gigabit Ethernet links between the OSTs and Alvarez.
3. Evaluation of Lustre scaling in conjunction with PDSF with GUPFS providing OST nodes.
4. Beta evaluation of the forthcoming Blue Arc OST storage devices.

InfinARRAY File System Evaluation

Maximum Throughput, primarily a storage system vendor, is developing the InfinARRAY IP network-based shared file system. This interesting and novel file system is based on federating storage processors (SPs) in a fashion that nearly eliminates metadata synchronization issues. It uses IP networks for data transfers and metadata operations, but does not have the NFS architectural limitations. The SPs are purely software implementations that run on Intel Linux systems. The GUPFS testbed now has adequate node resources to use some of them as SPs while retaining enough nodes as client systems to conduct an evaluation.

An interesting element of the InfinARRAY file system is that it will explicitly support Myrinet GM-based transfers. The small eight-port Myrinet switch obtained as part of the GUPFS testbed upgrade will be used to evaluate this capability. This Myrinet capability also opens the door to conducting testing of the InfinARRAY file system with a larger number of nodes by incorporating the GUPFS testbed Myrinet eight-port line card into the LBNL Alvarez cluster's Myrinet switch. This will allow the GUPFS testbed to provide eight Myrinet-connected SPs to the Alvarez cluster as integrated components of Alvarez itself, and as part of Alvarez's Myrinet fabric. This will allow scaling studies up to the full size of Alvarez (87 compute nodes) and with an increased number of SPs.

Contingent upon the availability of the InfinARRAY file system, the following activities and evaluations of InfinARRAY are planned for FY 2003:

1. Evaluation of the InfinARRAY file system in late Q2 using Gigabit Ethernet as the interconnect fabric.
2. Evaluation of the InfinARRAY file system in Q3 using Myrinet as the interconnect fabric, with both IP- and GM-based protocols.

Based on the outcome of the initial evaluation and the availability of resources and time, the following optional InfinARRAY evaluation activities will be attempted in FY 2003:

1. Evaluation of InfinARRAY scaling properties using the Alvarez compute nodes as clients, with GUPFS testbed nodes as SPs over a Myrinet interconnect between the SPs and Alvarez, utilizing both IP and GM protocols.
2. Evaluation of InfinARRAY scaling properties using the Alvarez compute nodes as clients, with GUPFS testbed nodes as SPs over Gigabit Ethernet links between the SPs and Alvarez.

GPFS File System Evaluation

IBM's GPFS file system is available on two systems at NERSC: the production Seaborg IBM SP system and the Intel-processor-based LBNL Alvarez Linux cluster. GPFS is currently not a candidate file system for center-wide deployment under the GUPFS project because of licensing issues and limited platform and OS support. Because of licensing and support issues, GPFS cannot be installed on the GUPFS testbed to conduct an evaluation. However, the presence of GPFS on multiple architectures at NERSC and its use on NERSC's primary production computing system provide a number of opportunities to the GUPFS project.

The GUPFS project intends to conduct an evaluation of GPFS on both the SP and the LBNL Alvarez Linux cluster. These evaluations will be conducted using the same general methodology

and benchmarks used to perform evaluations on the GUPFS testbed. Only those adaptations necessary to conduct a non-destructive, non-intrusive evaluation in a production environment will be made. Conducting these evaluations will enable the GUPFS project to:

1. conduct an evaluation of GPFS
2. compare the performance of GPFS running on two different architectures
3. examine the GUPFS project evaluation methodology and benchmarks on systems with a large number of nodes in order to gain insight into their ability to identify scalability problems and demonstrate/predict scalability
4. allow the comparison of GPFS with other file systems evaluated, based on the results from the Alvarez cluster when run at a scale similar to the GUPFS testbed.

The GUPFS project plans to conduct one GPFS evaluation on the SP, and at least one evaluation of GPFS on the Alvarez cluster during FY 2003.

7.2.2 Storage Technologies

A major objective for FY 2003 is to be able to demonstrate 1 GB/s sustained file system performance. The 1 GB/s (8 Gb/s) goal demonstrates that a global/parallel file system is capable of passing the 2 Gb/s performance bar that is currently set by the NFS/NAS servers.

The existing storage on the GUPFS testbed does not have adequate aggregate bandwidth to provide 1 GB/s sustained file system performance. Testing during the last year has demonstrated that the current storage provides only a fifth of the required aggregate bandwidth, even when used as dedicated local storage assigned to separate compute nodes. This is partially due to the 1 Gb/s Fibre Channel interfaces supported by the existing storage, and partly due to limitations in their controller performance.

In FY 2002, the GUPFS project acquired an EMC CX 600 to add to the GUPFS testbed. The selected CX 600 storage and controller not only support 2 Gb/s all the way to the disk, but also provide multiple 2 Gb/s Fibre Channel ports to access each of the controllers. They also provide the ability to configure the disks in multiple (four) independent internal storage streams. The total storage capacity (around 2 TB of RAID 5) was chosen to be relatively low in relation to the number of disk devices to ensure adequate to disk transfer bandwidth.

In FY 2003, we plan to use the newly acquired CX 600 to study different file system technologies with a focus on the aggregate performance and the scalability of the file systems.

7.2.3 SAN Fabric Technologies

The major focus of the GUPFS project in FY 2003 is to evaluate the performance characteristics of SAN fabric technologies and the interoperability of these technologies in a hybrid, multiple-fabric environment [22,23]. The ability of the file systems to operate in such a mixed environment is very important to the ultimate success of the GUPFS project. A picture of the current GFPFS fabric configuration is shown in Figure 17.

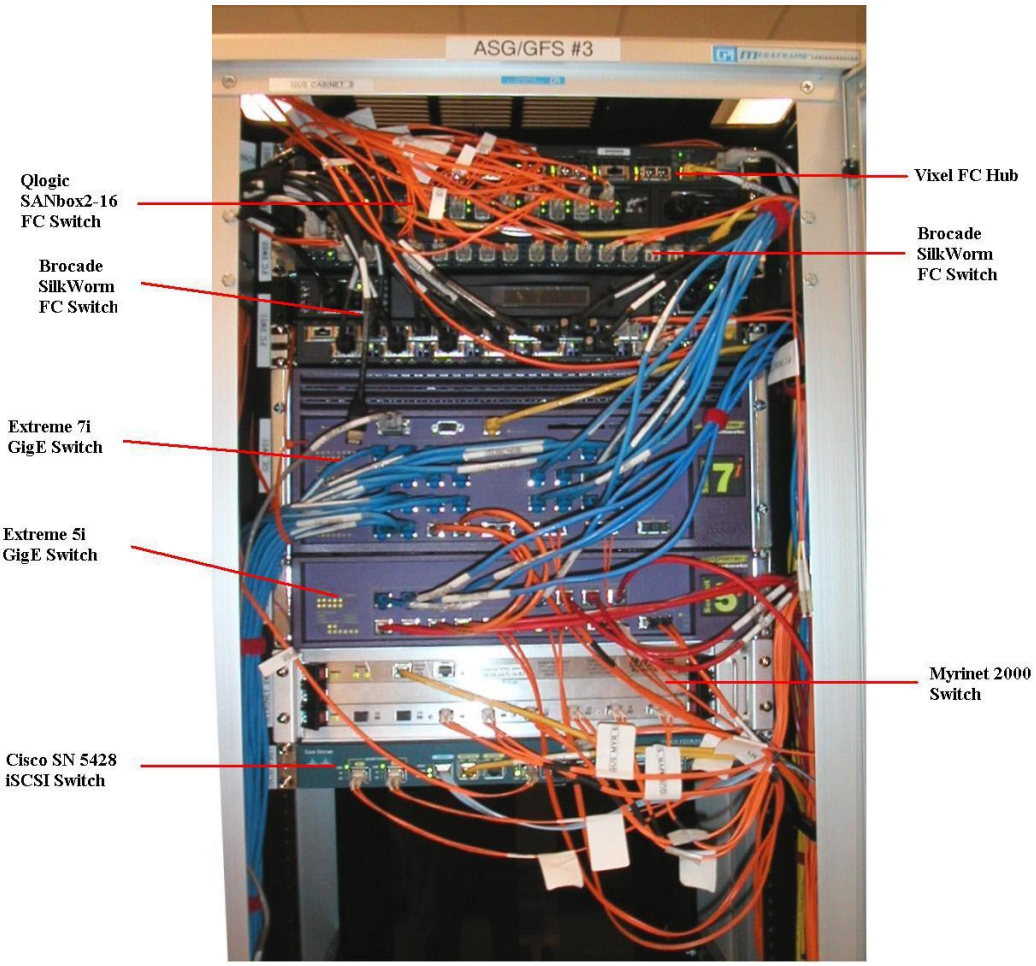

Figure 17. The GUPFS fabric configuration.

2 Gb Fibre Channel Switch

Fibre Channel (FC) technology has recently been upgraded from 1 Gb/s to 2 Gb/s bandwidth. Many FC vendors have introduced new switch products with better performance and higher port connectivity. This increase will allow substantially improved storage performance to be realized. It is important to evaluate the improved FC technology to determine how this may affect the fabric topology and the selection of the file systems.

SAN scalability is another important area for the near-term investigation when deploying new switches into existing fabric. SAN scalability measures how a fabric design can grow without requiring a substantial re-layout of existing fabric topology. An effective SAN architecture needs to be able to accommodate additional servers, switches, and storage with minimal impact to the existing SAN operation.

The existing GUPFS testbed currently has a Brocade SilkWorm 2800 16-port 1 Gb/s switch. In FY 2002, we added two additional switches to the testbed:

- Brocade SilkWorm 3800 16-port 2 Gb/s Fibre Channel switch
- Qlogic SANbox2-16 16-port 2 Gb/s Fibre Channel switch.

Both switches are capable of sustaining 32 Gb/s (full duplex) non-blocking switch throughput. During FY 2003, we plan to use these three switches to experiment with different fabric topologies and to evaluate the scalability and the performance characteristics of different topologies. We also plan to investigate how switches from different vendors may work together in a hybrid environment.

iSCSI Evaluation

Additional SAN fabric technologies are beginning to appear. Using Ethernet as a SAN fabric is now becoming possible due to the iSCSI standard for doing SCSI storage traffic over IP networks. This is very attractive as it allows lower-cost SAN connectivity than can be achieved with Fibre Channel, although with lower performance. It will allow large numbers of inexpensive systems to be connected to the SAN and use the cluster file system through commodity components.

To begin the iSCSI evaluation in 2002, we acquired five Intel IP Pro/1000 T IP iSCSI HBAs. The Intel iSCSI HBA can be used as an iSCSI initiator or an iSCSI target. We also acquired a Cisco SN5428 storage router to bridge between IP networks and FC SANs.

In FY 2003, we plan to evaluate the iSCSI technology in the following areas:

- iSCSI performance over traditional Gigabit Ethernet (GigE)
- iSCSI performance over Intel iSCSI HBA (Intel IP Pro 1000 HBA)
- the interoperability between IP routers, storage router, and FC switches.

We plan to compare the iSCSI performance and scalability against the native FC performance using Qlogic QLA2200 or QLA2300 HBA. We plan to measure the CPU overhead when the traditional GigE is used. We plan to also explore the possibility of using the Intel iSCSI HBA to implement a direct-attached iSCSI storage target to baseline the iSCSI protocol performance.

InfiniBand Switch

In addition to using Ethernet as a SAN, the newly emerging InfiniBand interconnect shows promise for use in a SAN as a transport for storage traffic. InfiniBand offers performance (bandwidth and latency) beyond that of either Ethernet or Fibre Channel, with even higher bandwidths planned. With the promise of commodity-level pricing in the future, this is a technology that needs to be studied, even in its very early stages. As with Ethernet fabrics, an important part of the InfiniBand technology that needs to be examined is fabric bridges between InfiniBand and Fibre Channel SANs. Another area of InfiniBand technology that needs to be explored is storage transfer protocols (e.g., SRP) and methodologies. In 2002, we expanded the GUPFS testbed to include an InfinIO 7000 Starter Kit from InfiniCon Systems to begin InfiniBand evaluation. Figure 18 shows a picture of the InfinIO 7000 switch.

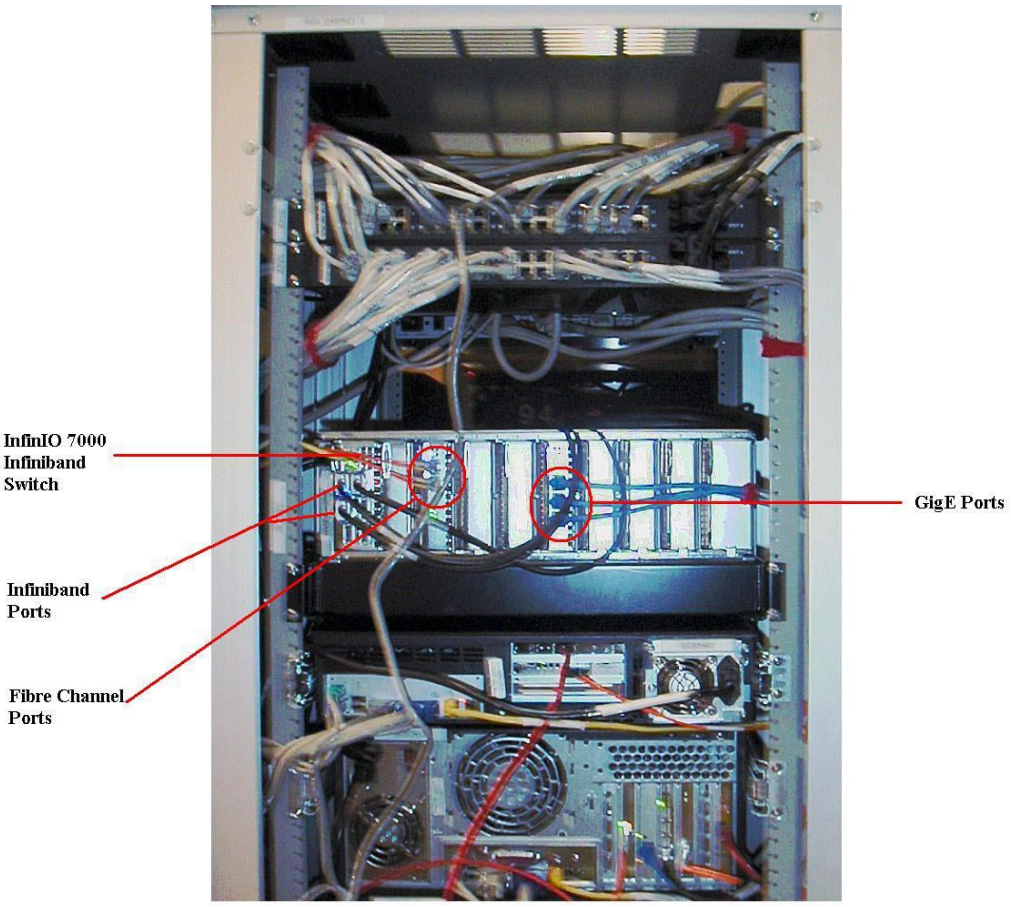

Figure 18. The InfinIO 7000 shared I/O switch.

The InfinIO 7000 shared I/O system is a multi-protocol networking system for shared I/O and InfiniBand switching. The server nodes attach to the InfinIO 7000 switch via a high-speed, 10 Gb/s (4x) InfiniBand connection to access a pool of virtual I/O resources, including Fibre Channel SANs, Ethernet SAN or NAS, and native InfiniBand fabrics. This shared I/O architecture eliminates the need for individual NICs, HBAs, and cabling on the server nodes. This saving of operational and capital costs for infrastructure could be significant.

Each InfinIO chassis supports dual 4x InfiniBand switch modules and up to eight I/O "personality modules." The I/O personality module can be a three-port 1 Gb Virtual Ethernet Exchange (VEx) card, a two-port 2 Gb Virtual Fibre Channel Exchange (VFx) card, or a six-port InfiniBand Expansion (IBx) card. Chassis slots can be populated with any mix of personality modules, which can be hot-swapped to accommodate configuration changes. The starter kit we acquired for the GUPFS project includes three 1x Intel InfiniBand HCAs and an InfinIO 7000 switch. The InfinIO switch is populated with a six-port IBx, a two-port 2 Gb VFx, and a three-port 1 Gb VEx. The three HCAs are currently installed on two Linux servers and one Windows 2000 server. The Windows 2000 server is required for the InfiniBand subnet management using the subnet management software from Lane15 Software. The Lane15 subnet management software is currently available only on Windows 2000, but a Linux port will be available shortly.

In FY 2003, we plan to use the InfinIO 7000 starter kit to evaluate the InfiniBand and shared I/O technology in the following areas:

- iSCSI performance over InfiniBand and VEx vs. iSCSI over Intel Gigabit Ethernet NIC
- I/O performance over InfiniBand and VFx using SRP vs. I/O over Qlogic QLA2340 HBA
- the interoperability between InfiniBand switch, IP routers, storage router, and FC switches.

InfiniCon Systems is also in the business of manufacturing 4x InfiniBand HCAs. We have signed up to be a beta customer of the new 4x HCA once they are available. The new HCA will give us much higher bandwidth (10 Gb/s) than the 1x HCA (2.5 Gb/s) that we currently have. We plan to repeat the I/O benchmarks using iSCSI, SRP, and native FC interface with the 4x HCAs. We would also like to study InfiniBand scalability. However, we may be limited by the number of HCAs, IBx, VFx, and VEx cards that we can obtain. We plan to explore any possible collaboration opportunities with other InfiniBand vendors for additional testing and evaluation of the InfiniBand technology.

High-Speed Interconnect

The GUPFS project is also working with Maximum Throughput to conduct evaluations of their InfinARRAY file system. This file system uses IP networks for data transfers and metadata operations, but does not have the NFS architectural limitations. One interesting element of the InfinARRAY file system is that it will explicitly have Myrinet GM-based transfer capability. To support testing this and allow expansion of testing to a large number of nodes by directly connecting to the LBNL Alvarez cluster's Myrinet fabric, a small Myrinet switch was acquired. This will allow testing of the file systems over high performance interconnects using both IP and, when possible, GM local to the GUPFS test bed and by moving the Myrinet line card into the Alvarez switch, providing file system services to the Alvarez compute nodes.

7.3 Longer-Term Investigations and Activities

While the GUPFS project remains focused on conducting near-term investigations, it is also planning and preparing for other investigations and activities that need to be conducted over the longer term. Many of these activities relate to preparing for the second phase of the GUPFS project, while others involve continuation of the evaluation process and beginning to address additional functionality that is required for a successful deployment.

The longer-term investigations and activities that the GUPFS project is planning include:

- Additional technology evaluations from all component technology areas. Which additional technology evaluations will conducted depends on the outcome of the near-term evaluations and on future developments in the component technology areas.

- Integration of HPSS with various shared file systems. This ambitious activity is focused on incorporating hierarchical storage management (HSM) functionality into the most promising candidate shared file systems through the XDSM/DMAPI interface and capability of HPSS. This will require work not only in the candidate file systems, but also in HPSS. It may also involve integration of direct HPSS movers with the shared file systems.

Lustre is an obvious target for such work, as is StorNext. ADIC is very interested in integrating its DMAPI interface with HPSS. Even more extensive integration of HPSS with the shared file systems is also being explored, although it is probably beyond the direct scope of the GUPFS project.

- Investigations and evaluations of different logical volume management (LVM) software for use in virtualizing and presenting storage devices to the shared file systems. This includes both file system client host level software, and virtualization at the storage device and fabric level. Different shared file systems may require different or specific LVMs.

Some of the planned activities relating to the second phase of the GUPFS project are:

- Retesting of the most promising technologies, particularly the file system technologies, both to reassess them and to evaluate their progress and continued suitability in light of future developments.

- Narrowing the selection of candidate technology components in preparation for the final selection of the technologies to be deployed.

- Planning for the consolidation of disk storage for the production computational and support systems, including organizational arrangements for supporting centralized storage and a center-wide shared file system, and procurements.

- Initiating planning of the phased rollout and deployment of the center-wide shared file system and consolidated, centrally managed storage.

Much remains to be done in both the near term and longer term to ensure a successful conclusion to the GUPFS project. With careful planning, diligent effort, and continued technological advances, particularly in the file system arena, this should be an achievable outcome.

Appendix A
Benchmark Code Descriptions

A.1 MPTIO Benchmark

MPTIO is a parallel and multi-threaded I/O benchmark code used to measure the aggregate I/O capability of a parallel file system and the underlying raw storage network and device. It uses MPI to coordinate and synchronize the I/O operations across processes running on multiple file system or device clients. It is *not* an MPI-I/O application; it uses Portable Operating System Interface (POSIX) read/write system calls to perform all the I/O. Each process spawns multiple worker threads that perform the actual I/O operations. The main or master thread is used only to coordinate the work and communicate with the other MPI processes. The worker threads have the effect of emulating asynchronous I/O; each independently opens the file or device, performs the I/O and closes the file. All processes communicate timing results back to process zero, which computes and reports individual and aggregate I/O rates to an output file. MPTIO was adapted from a heavily modified version of TIOTEST, a single-process multi-threaded I/O benchmark written by Mika Kuoppala and distributed under the GNU Public License.

```
MPTIO   [-G|-P|-T|-R dev [-R dev]…] [–f fsize | -F fsize]
        –b recsz –t nth [-k num [-k num]…] [-N name]
        [-d dir] [-U] [-O p] [-O t] [-a nrand] [-r]
        [-I] [-s offset] [-K] [-U]
Where:
   -f fsize The per-process file size (in Bytes).
   -F fsize   The global file size (in Bytes)
   -b recsz The record size of each I/O operation.
   -t nth     The number of threads per process.
   -G         I/O to a single, Global file.
   -P         I/O to a per-Process file.
   -T         I/O to a per-Thread file.
   -R dev     I/O to the given block device.
   -k num     Test "num" should not be run.
   -N name    The base name of the output file.
   -d dir     The pathname of the working directory.
   -U         Build a Unique working directory.
   -a nrand   The number of I/O ops per random test.
   -r         Rotate file segments after each test.
   -I         Use Direct I/O (if available).
   -O (p|t)   Process and/or thread overlap
```

```
-s offset  Start all I/O at the given offset
-K         Keep files. Don't unlink at end of run.
-C         Check files between tests.
NOTES:
```

- Only one of the –G, -P, -T, or –R options may be used. The default is '-G'. Multiple instances of '-R dev' may be used in a run to specify that I/O should be done to multiple raw or block devices.
- Only one of the –f or –F options may be used. This is a convenience, in some situations it is more natural to specify the per-process file size (-f) and in others it is more natural to specify the global file size (-F).
- The size parameter of the '-f', '-F', '-b' and '-s' options is specified in bytes, and may have the form nnnX where nnn are digits and 'X' is either 'K' to specify Kilobytes, 'M' to specify Megabytes or 'G' to specify Gigabytes. That is, 64K is equivalent to 64*1024 = 65536 bytes.
- The size parameter of the '-f' (and, by extension the '-F') option is the approximate amount of I/O each process will perform. The actual value will be adjusted as described below.
- The number of processes created and where they will run is specified outside of MPTIO, by the MPI runtime system.

MPTIO was designed to be flexible in the way the threads and processes are assigned the work of performing the I/O on the file(s) or device(s). By default, each thread of each process is assigned a disjoint (non-overlapping) contiguous region of one of the files or devices upon which it performs various I/O operations. If the process overlap flag is specified (**`-O p`**), then all the processes performs I/O over the same region of the file or device, but the threads within a process act on disjoint subregions. If the thread overlap flag (**`-O t`**) is specified, then all threads within a process perform I/O over the same region. If both overlap flags are specified, then all threads perform I/O over the same region of a file or device. The only exception is if multiple devices are specified; then the threads of each process are divided equally among the devices.

I/O can be performed either on files in a file system or directly to a block device or using Linux raw I/O to a device. If the **`-G`** option is specified, all threads perform I/O to a single file in a cluster file system. If the **`-P`** option is specified, all threads within a process perform I/O to the same file, but each process acts on a different file. If the **`-T`** option is specified, each thread of each process will perform I/O to a separate file. If, instead, the **`-R dev`** option is specified, then all I/O is done to the specified block or raw device. In this case, the record size must be a multiple of the device block size and all I/O must be block aligned. MPTIO adjusts input parameters to insure this is the case. Further, one can specify multiple devices on the command line (multiple instances of **`-R dev`**). MPTIO adjusts the number of threads per process such that an equal number of threads is assigned to perform I/O on each device. This feature was added to compare the effects of raw I/O to multiple devices with raw I/O to a logical block device, which stripe across the underlying devices.

As mentioned above, each thread is assigned a contiguous region of a file or device to which it performs I/O operations. In particular, each thread is assigned four parameters that specify the region over which it performs I/O:

- file or device pathname
- byte offset within file or device
- I/O record (or block) size (**`recsz`**)
- number of records within the region (**`numrec`**).

In reality, **`recsz`** and **`numrec`** are identical for all threads. These parameters are determined by various command line arguments. **`Recsz`** is specified as a command line argument, but might be adjusted in the case of block or raw I/O. The pathnames are specified directly on the command line in the case of block or raw I/O, but are generated by MPTIO in the case of file I/O. In the latter case, the files are created in the directory specified by the **`-d dir`** argument. This directs the I/O to be performed in the file system containing this directory. Clearly, the pathname assigned to a particular thread is dependent on the **`-G`**, **`-P`**, **`-T`** or **`-R`** options as outlined above. The per-thread byte offset and **`numrec`** values are determined by a complicated interaction of the number of threads, the per-process **`filesize`**, and whether the process or thread overlap options are selected. The per-process **`filesize`** is considered a suggestion by the user and may be adjusted so that: **`filesize = nthread * numrec * recsz`**. That is, **`numrec`** is selected so that the expression on the right most closely approximates the user's **`filesize`** specification. Also, by default each process works on a different file or on a different region of a file or device. For example, Figure A-1 illustrates how four processes, each with two threads, divide a single device or file.

Process 0		Process 1		Process 2		Process 3	
Thrd 0	Thrd 1	Thrd 0	Thrd 1	Thrd 0	Thrd 1	Thrd 0	Thrd 1

Figure A-1. How four processes, each with two threads, divide a single device or file.

If the thread overlap option (**`-O t`**) is selected, then all threads in the process work on the same file region and **`numrec`** is selected so that **`filesize = numrec * recsz`**. In this case, duplicate I/O is performed as shown in Figure A-2.

Process 0	Process 1	Process 2	Process 3
Thrd 0	Thrd 0	Thrd 0	Thrd 0
Thrd 1	Thrd 1	Thrd 1	Thrd 1

Figure A-2. Duplicate I/O with the thread overlap option.

Clearly, this option is incompatible with the **`-T`** option. If the process overlap option (**`-O p`**) is selected, then all processes work on the same region of the same file or device. This option is incompatible with the **`-T`** or **`-P`** options.

The **-s offset** option causes all threads to add this offset value to their assigned offset location. This option can be used to flush a controller cache. Writing or reading at an offset much larger than the intended I/O region causes the cache to be overwritten. A subsequent I/O test acting on different disk blocks does not see cache effects.

By default, each execution of MPTIO performs a series of five I/O tests on the underlying file(s) or device(s). They are:

- Test 0: Sequential Write Test
- Test 1: Read-Modify-Write Test
- Test 2: Random Write Test
- Test 3: Sequential Read Test
- Test 4: Random Read Test

In the sequential write test, each thread writes its segment of its file or device in sequential order. In the random write test, each thread performs write operations to random records within its section of the file or device. That is, it seeks to write a random record within its section and then writes the buffer. The number of random write operations is specified by **numrec**, but may be changed by using the **-a nrand** command line option. The random I/O tests can have low performance because they generally defeat the file system or device caching algorithms, and therefore it is useful to reduce the number of operations using this option. In the Read-Modify-Write test, each thread reads the next record of its segment, modifies a word in the record, then seeks back to the beginning of the record and rewrites it. The read tests (3 and 4) are similar to the write tests. The **-k num** command line option causes the specified test to be skipped. That is, **-k 1 –k 2 –k 4** specifies that each thread should only perform the sequential write test, followed by the sequential read test.

When performing file I/O, the file or files used in the test are created (by **Test 0**) in the directory specified by the **-d dir** command line option. By default, **dir** is the current working directory. In addition, if the **–U** option is specified, the files are created in a unique subdirectory of the working directory so that a series of tests may be run without overwriting the test file. If the **-K** command line option is given, the test tiles are kept after the job completes. The default behavior is to unlink the files at the end of the test. Note that when doing file I/O, **Test 0** should not be skipped unless the test files already exist from a previous run. The remaining tests in the suite depend on the files **Test 0** creates. When doing I/O to a device, **Test 0** may be skipped since we can always read (garbage) from a device.

The **-r** command line option specifies that each thread "rotates" its file segment information (its file or device name and file offset) to the corresponding thread of the process with one lower rank. That is, thread **K** of process **P** rotates its file and segment info to thread **K** of process **(P-1) (mod NUMPROC)** and performs I/O on a different file segment in the upcoming test. This operation is performed after each test in the sequence. This can be useful to eliminate or reduce the effects of on-node file system caching between tests.

The **-I** option enables direct I/O, assuming the underlying file system supports this feature. That is, each thread opens its file with the **O_DIRECT** flag. Where supported, direct I/O causes the I/O

operation to bypass the system buffer cache. In this case, the I/O operations must be a multiple of, and aligned on sector boundaries.

By default, process zero writes the results of the tests to standard output. The **-N basename** option causes the report to be written to a date and time-stamped filename of the form **basename_ YYYYMMDD_HHMMSS** where **YYYY** is the year, **MM** is the month, **DD** is the day of the month, **HH** is the hour, **MM** is the minute, and **SS** is the second. This file contains a collection of information including process-specific and aggregate I/O rates as illustrated below:

```
Size (MB)  Recsz (KB)  Num_proc  Thrd/proc  Tot_thrd  Rec/proc  Rec/th
---------  ----------  --------  ---------  --------  --------  ------
   200         256         3          4         12        800      200

Test Name   Recsz  Tot_thrd  Rate MB/s    E-Rate    E-Time    %DtVar
----------  -----  --------  -----------  --------  --------  --------
Seq_Write   256       12          9.17       9.17     65.44      1.09
Seq_Read    256       12         88.43      88.43      6.79      9.44

NODE DETAILS
|------|----------------------|----------------------|
| Node |  Seq_Write           |  Seq_Read            |
|      |  Rate  Time   %Sys   |  Rate   Time  %Sys   |
|------|----------------------|----------------------|
|  0   |  3.08  64.90  5.82   |  29.87  6.70  33.60  |
|  1   |  3.06  65.44  5.84   |  32.59  6.14  34.87  |
|  2   |  3.09  64.73  5.81   |  29.48  6.78  28.59  |
|------|----------------------|----------------------|
```

This test was run with three processes, each containing four threads performing the sequential read and write tests. The first section lists the run-specific parameters; the second section reports the aggregate I/O rates for each test; and the third section lists the per-process or per-node details. Each process reports to **Process 0** the amount of I/O it performed in each test, how long it took, and the percent of system time recorded by the kernel during the test. **Process 0** then generates the individual I/O rates and computes an aggregate rate based on the total amount of I/O during the time period. Note that two aggregate rates are reported. The first uses the maximum time reported by any of the processes; the second (**E-rate**) uses the elapsed time between the barrier that signals all processes can run and the barrier that signals all processes have completed. The **DtVar** value is the percent of variation between the individual timings and the elapsed time. If this value is large, the individual processes took substantially different amounts of time to complete the test. A Perl script named **mptsum.pl** was written to parse the output of MPTIO and generate less verbose summary information.

Example 1:
```
mpirun -np 4 MPTIO -G -f 1G -b 256K -t 2
-d /mnt/gfs/testdir
```

This script creates four MPI processes, each of which creates two worker threads to perform the I/O. Each process generates 1 GB of data split equally between the two threads. The I/O is performed in records of 256 KB. All threads read and write to non-overlapping sections of a single

global file. The 4 GB file is created in the **`/mnt/gfs/testdir`** directory. Figure A-3 is a diagram of the per-thread sections.

Figure A-3. Diagram of per-thread sections for Example 1.

Example 2: `mpirun –np 2 –P –F 512M –b 64K –t 4 –r –k 1 –k 2 –k 4`

This script creates two MPI processes, each with four threads. Only the sequential write and sequential read tests are performed, but the per-thread file and segment information is exchanged between the two processes after the sequential write test. The test generates two files, each with a size of 256 MB for a total of 512 MB (note the **`-F`** option). Each thread is responsible for a 64 MB segment of its respective file. The I/O is done in units of 64 KB. Figure A-4 is a diagram of the files and the thread sections.

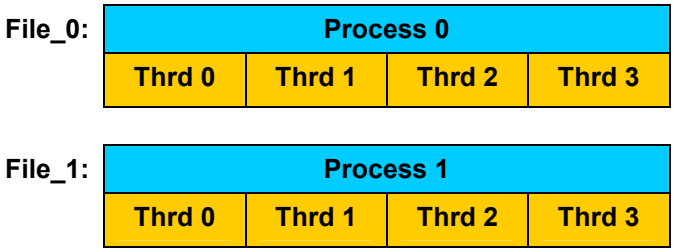

Figure A-4. Diagram of the files and thread sections for Example 2.

Note that because of the **`-r`** option, after the sequential write test, **`Process 1`** works on **`File_0`** during the sequential read test, and **`Process 0`** works on **`File_1`**.

Example 3: `mpirun –np 3 –F 512M –b 64K –t 2 –R dev0 –R dev1`

In this case we have three processes, each with two threads performing I/O on two devices. **`Thread 0`** from each process is assigned to **`dev0`** and **`Thread 1`** is assigned to **`dev1`**. The total **`filesize`** is 512 MB, and so the per-process **`filesize`** is about 170.6 MB divided between the two devices, or about 85.3 MB per process per device. Each thread does I/O in record sizes of 64 KB with **`numrec`** = 1,365 records.

A.2 METABENCH Benchmark

METABENCH is a prototype parallel file system metadata benchmark code. Its intended use is to measure various metadata performance characteristics of a parallel file system by synchronizing the operations across multiple processes running on different clients of the file system. METABENCH uses MPI to coordinate the operations and to collect the results.

```
Metabench       [-CSUATk0 [-c files] [-b size] [-o name]
                [-w dir] [-f freq]
Where:
    -C          Run file creation tests.
    -S          Run file stat tests
    -U          Run file utime tests.
    -A          Run file append tests
    -T          Run untar test.
    -c nfile    Number of files per test
    -b nbyte    Number of bytes used in append test.
    -f freq     Frequency of reporting file creation.
    -o name     Base name of output file
    -w dir      Path to working directory
    -k          Cleanup. Remove test directories.
    -0          Allow process 0 to participate.

NOTE: The number of processes used in the test and where they will run
is specified to the MPI runtime system
```

Let **nproc** represent the number of processes used in the MPI job. In most of the tests, **Process 0** is used only to create the test directories and files, and to collect and report the results. The reason for this is that many parallel file systems cache metadata information on the node that opened, created, or modified the file most recently. In order to give all processes equal and fair access to the metadata, we use **Process 0** to perform the test setup operations and do not allow it to participate in the actual test. If **nproc = 1** or the **-0** command line option is used, **Process 0** participates in the tests.

All tests are conducted in subdirectories of the working directory specified by the **-w dir** command line option. By default, **dir** is the current working directory.

If the **-C** command line option is specified, the file creation tests are performed. Note that all files created by these tests have zero length. The tests are:

- **Timing file creation rates.** After **`Process 0`** creates the test directory, process **`nproc-1`** enters the directory and records how long it takes to create the number of files specified by the **`-c nfile`** command line option. The files are created in batches specified by the **`-f freq`** command line option and the per-batch timings are reported. For example, **`-c 20000 -f 1000`** creates 20,000 files in a test directory, reporting how long it takes to create the first 1,000 files, then how long it takes to create the second 1,000 files, etc. Some file systems store directory entries in a linear list, with the result that file creation and loop-up rates grow linearly with the number of entries. In such cases, the performance degrades rather dramatically as the size of the directory grows. Other file systems store the entries in a hash table or binary tree, with the result that file creation and loop-up rates remain constant or grow as the log of the number of entries. This test determines the file creation rate as the number of entries in the directory grows. The results can be used to differentiate the effects of creating large directories from the effects of lock contention in the parallel file creation tests.

- **Parallel file creation in separate directories.** In this test, **`Process 0`** creates a subdirectory for each of the processes participating in the test. Each process enters its directory, waits on a barrier, then creates a specified number of files. Each process records how long it takes to create the files and sends this information to **`Process 0`**. The number of processes participating, **`N`**, is **`nproc-1`**, unless the **`-0`** command line option is specified, in which all processes participate. The number of files created by each process is **`nfile/N`**.

- **Parallel file creation in one directory.** In this test, **`Process 0`** creates a single subdirectory, and all participating processes then enter it. After a barrier, each process measures the amount of time it takes to creates **`nfile/N`** files. The results are collected by **`Process 0`** for reporting.

If the **`-S`** command line option is specified, the **`stat`** tests are run:

- **Parallel file stats in separate directories.** In this test, **`Process 0`** creates a subdirectory for each of the participating processes and populates each with **`nfile/N`** files. Then the processes enter their respective directory, and after a barrier, record the amount of time it takes to stat each of the files in the directory. The results are collected on **`Process 0`** for reporting.

- **Parallel file stats in one directory.** In this test, **`Process 0`** creates a single subdirectory and populates it with **`nfile`** files. Then each participating process enters the directory, opens it and walks to the **`k*(nfile/N)`** entry, where k is the relative rank of the process in the participating set. After a barrier, each process then stats the following **`nfile/N`** entries. That is, each process stats a different subset of the files in the directory. Once again, the time to stat the files is collected on **`Process 0`** for reporting.

If the **`-U`** command line option is specified, the **`utime`** tests are run. These are identical to the **`stat`** tests except that rather than stating the file, its access and modification time is updated using

the **utime** system call. Note that the **stat** tests read the metadata, whereas the **utime** tests modify the metadata.

If the **-A** command line option is specified, the **append** tests are run. Again, these tests are identical to the **stat** and **utime** tests except that in this case the files are appended by **nbyte** bytes. In these tests, not only is the metadata modified, but file blocks must be allocated to hold the appended data.

If the **-T** command line option is specified, the untar test is run. In this test, **Process 0** creates **N** subdirectories and each participating process enters its respective subdirectory. After a barrier, each process untars a UNIX tar file within that directory. The amount of time the process takes to untar the file is collected on **Process 0** and recorded. The tar file must be the same on each client of the parallel file system and should be held in a local (fast) directory, such as **/tmp**. Currently, the tar file used is **glibc** version 2.3. It contains 9188 files and 459 directories for a total of 75.8 MB of data.

By default, **Process 0** writes the individual and aggregate metadata rates to standard output. If the **-o basename** command line option is provided, the results are written to a file with the date and time-stamped name: **<basename>_YYYYMMDD_HHMMSS**. Here, **YYYY** is the year, **MM** is the month, **DD** is the day of the month, **HH** is the hour, **MM** is the minute, and **SS** is the second.

If the **-k** command line option is provided, **Process 0** performs cleanup after each test completes. That is, it removes the files and directories created during the test. This can be a time-consuming operation.

A.3 Additional I/O Benchmarks

This section lists additional I/O benchmarks that have been or will be used during the evaluation. Some benchmarks may be specific for file system evaluation and may not be directly applicable for the storage evaluation.

NOTE: The list of benchmarks is still evolving. Additional benchmarks may be developed or obtained from other sources and added to this list.

A.3.1 Bonnie

Bonnie is an I/O benchmark to measure the performance of UNIX file system operations. Bonnie runs a series of tests and measures the elapsed time per operation as well as the CPU overhead. The tests are conducted on a file of known size (100 MB by default) and the term *chunk* refers to the read/write unit size for block I/O operations. The default chunk size is 16 KB. The operations are:

1. The file is created and written sequentially using **putc**, then closed.
2. Open file from step 1, **read** each chunk, dirty one word in the chunk, **lseek** back to the start of the chunk, and re-**write**. Close the file after completion.
3. Delete the file and re-create it with block **write**s of size chunk.
4. Open the file created in step 3 and read it using **getc**.

5. Reopen the file created in step 3 and **`read`** it using block reads.
6. Fork **`N`** (default = 4) children and have each open the file created in step 3, then have each perform a series of **`lseek`** operations to random locations in the file. The idea is to make sure there is always an **`lseek`** operation pending.

A.3.2 Tiotest and Tiobench

Tiotest stands for Threaded I/O Test. This program creates a specified number of threads (at system contention scope) that perform I/O on different files, possibly in different directories or file systems. The threads are tightly synchronized so that each is activated at the same time on an SMP node. The I/O can be done using either **`lseek`**, **`read`** and **`write`** or via a memory-mapped file. The tests consist of:

- **`do_write_test`**: Open the file and optionally **`seek`** to a starting location. Data is written to the file in block sizes specified on the command line. Each write is individually timed **`gettimeofday`**, and statistics, such as largest latency and average latency, are computed.
- **`do_random_write_test`**: Similar to **`do_write_test`** except that each write operation is proceeded by a **`seek`** to a random location (block-size aligned) in the file.
- **`do_read_test`**: Similar to **`do_write_test`** except that it reads from the file (created in **`do_write_test`**) and optionally checks the data returns by computing a CRC (cyclical redundancy check) value and comparing with what it should be.
- **`do_random_read_test`**: Similar to **`do_random_write_test`** except that it reads blocks from the file and optionally checks the data for consistency.

Each test is timed. In addition, each operation is timed individually, in order to compute maximum latency and to compute average latency.

The code also comes with a Perl script named **`tiobench.pl`** that makes it easy to run a series of tests, with different block sizes, etc. It also nicely formats the output.

The advantage of Tiobench over Bonnie is that tests can be run on a variety of block sizes without having to recompile the code. This code only tests block I/O and not **`getc`** and **`putc`** from the **`stdio`** library.

A.3.3 Iozone

Iozone is another I/O benchmark with a lot of options to control how the test is run. The range of options includes what to include in benchmark timings, what options to **`open`** (e.g., **`O_SYNC`**), whether to use **`mmap`**, **`read`**, or **`write`** to perform the I/O, whether to double buffer, whether to purge the processor cache between operations, whether to use processes or threads to generate the workers, and many more. It even includes an option to unmount and remount the file system between sub-tests to force a flush of the buffered data and metadata. In addition, you can set the file sizes, the read/write record sizes, stride patterns for strided I/O, and others. The output can be written to a text file or to a file that Excel can read directly into a spreadsheet for graphing purposes.

By default, the following series of tests is performed:

1. **Write:** Create a file of the specified size then open it for writing. Write data to the file in the specified record size, and in sequential order. Close the file.
2. **Rewrite:** Reopen the file created in the Step 1. Rewrite each record with the specified record size and in sequential order. Close the file.
3. **Read:** Open the file created in step 1, read the first byte, then seek back to the beginning of the file to prime the instruction and translation lookaside buffer (TLB) cache. Then read the file sequentially with the specified record size. Finally, close the file.
4. **Reread:** Identical to the test in step 3, only it is performed just after step 3 and if the file is not too large, there may be blocks already cached in memory.
5. **Random read:** Open the file created in step 1. Loop over the number of records in the file and for each, compute a random offset (aligned on a record boundary), seek to that location, and read the record. Close the file.
6. **Random write:** Same as step 5, except write the record rather than read it.
7. **Backward read:** Open the file created in Step 1, seek to the end minus the length of the selected record size. Walk backwards through the file, reading the record then seeking backwards two record lengths (one for the record just read, and one to backup to the previous record). Close the file.
8. **Record rewrite:** A temporary file is created and opened. Loop over the number of records in the original file (created in Step 1) and for each: write a record to the temp file, then seek back to the beginning. That is, rewrite the same record over and over again, the number of times determined by the number of records in the original file.
9. **Strided read:** In this step, the records of the file are read, not sequentially, but with a constant stride, wrapping back to the beginning of the file in a periodic fashion. The goal is to read each record of the file so that when the wraparound occurs, the initial record is selected to the next that has not already been read.
10. **Fwrite:** This is similar to Step 1, except that the **`stdio`** functions **`fopen`** and **`fwrite`** are used. That is, it tests the performance of the standard I/O library.
11. **Frewrite:** Same as Step 2 only using the **`stdio`** library.
12. **Fread:** Same as Step 3 only using the **`stdio`** library.
13. **Freread:** Same as Step 4 only using the **`stdio`** library.

A.3.4 Pioraw

Pioraw is an MPI program in which each node attempts to measure the amount of raw I/O it can perform from each of the given processes in the MPI job and within a given time period. The experiment is repeated for varying block sizes, and aggregate I/O volume and aggregate bandwidth are reported for both read and write performance.

Each node is writing to a separate file, so there is not metadata contention. The point is to find when the I/O system saturates, thereby giving an indication of the maximum I/O throughput for the entire parallel system.

A.3.5 Mdmark

Mdmark is a metadata benchmark written by Adrian Wong at NERSC. It constructs a directory hierarchy based on parameters gotten from scans of actual NERSC user file systems. It then runs a series of metadata stress tests on the directory hierarchy and reports results in terms of number of metadata operations per second.

First, Mdmark creates a directory hierarchy based on profile information gathered from real user file systems, such as one of the home file systems on the NERSC IBM SP. The profile information is a set of probability density functions that characterize file size and the distribution of files and subdirectories at each level.

When the directory structure is created, a random number generator is used to extract from the PDF the actual number of entries, subdirectories, and file sizes to be created. The process is deterministic if the same initial seed (to the random number generator) is used. That is, if the same seed is used, the same structure is built, otherwise a different hierarchy is constructed. The file and directory names are randomly generated and, as such, may not have the same lookup characteristics as actual user-generated file names. Input parameters can be used to limit the size of the directory hierarchy. The time it takes to create the directory structure, and the number of entries created, is reported to standard output.

Once the directory structure is built, other optional scans of the structure can be performed. Each scan is timed and reported to standard output. The options are:

- **Scan:** Walk the directory structure and stat each entry.
- **Scan and update**: Walk the directory structure, stat each inode, and update the access and modification times of each inode.
- **Scan and delete**: Walk the directory structure, stat each inode, and delete the files and subdirectories as you go. It recursively deletes all the regular files in the directory first, then removes the subdirectories.
- **Generate stats**: Walk the directory structure and gather statistics on the number of entries and number of subdirectories at each level. It produces a histogram of this information overall and by level.

If selected on the command line, a mix of metadata operations is performed on the directory structure by a collection of subprocesses forked by the main program. Each process scans the directory structure and randomly performs metadata operations such as stat, update access, and modification times, change directories up and down, create and delete files and subdirectories, etc. Counters are kept to record the number of metadata system calls performed, and the totals are written to standard output. The number of processes and length of time they run are controlled by command line parameters.

Appendix B
Current Results Details

B.1 QLOGIC 2340 HBA Performance

The purpose of this test is to demonstrate the best single-HBA performance (bandwidth in MB/s) of the Qlogic SANblade 2340 host bus adaptor.

The 2300 series HBAs have integrated a RISC (reduced instruction set computing) processor, the Fibre protocol engine and transceivers into a single Fibre Channel controller chip. The single chip integration increases reliability and performance with less CPU utilization. The 2300 series HBAs offer 2 Gb/s performance, are available in PCI-X form factor, and are backwards compatible to PCI. The 2300 series HBAs have been designed to have optimal performance in SAN and cluster environments.

The tests included single-stream and multi-stream tests with different I/O sizes.

B.1.1 Linux Host Configuration

The test host is a Linux platform with the following configuration:

- SuperMicro P4DP6 motherboard Intel E7500 chip
  - two Intel P4-2.2GHz Xeon processors
  - 2 GB DDR PC2100 ECC memory
  - two 64-bit 100 MHz PCI-X
  - four 64-bit 66 MHz PCI
- RedHat 7.3 with 2.4.18-5 kernel
- One Qlogic QLA2340 2 Gb HBA sitting on a PCI-X slot with a modified Qlogic v6.1b2[4] driver

B.1.2 Storage Device

The storage used for the HBA performance test is an EMC CX 600 with the following configuration:

- Three DAE (disk array enclosure) trays, each with 15 10K RPM 73 GB disks.
- Two SPE (storage processor enclosure) controllers: SPA (storage processor A) and SPB (storage processor B). Each SP has four 2 Gb host ports. The test used only one port.
- 4 GB cache per SPE: read and write caching was enabled.
- Read cache: SPA = 1470 MB, SPB = 1470 MB.
- Write cache: 2,000 MB (mirrored).
- A single five-disk Raid 5 LUN, with five disks from the same tray, was configured for the test.
- Prefetch = 512
- CX600 was attached directly to the test host; no switch was used.

[4] The Qlogic v1.6b2 driver was modified to avoid a known performance issue caused by bounce buffers (see http://www.tldp.org/HOWTO/IO-Perf-HOWTO/overview.html for more details). See also next section for the I/O performance with and without the bounce buffer.

B.1.3 Test Scripts and Setup

We used the **lmdd** program from the Lmbench benchmark kit[5]. The **lmdd** program copies data blocks from a specified input (file or raw device) to a specified output (file or device). It prints out the timing statistics after completing, which is useful for timing I/O. We used the raw utility (see **man raw**) to bind a raw device to the test device for testing. The following are some examples of single-stream reading and writing tests:

- Read: `lmdd if=/dev/raw/raw10 of=internal bs=32k count=32768`
- Write: `lmdd if=internal of=/dev/raw/raw10 bs=32k count=32768`

The test was set up to measure the best possible read and write performance from the host HBA to the cache of the controller on the storage array. To accomplish this, we have limited our file size to 1 GB, which is a little bit less than the read and write cache size.

We also changed the block sizes in the tests to see how the port performance may be affected by the I/O size. The block sizes used include 32 KB, 64 KB, 256 KB, and 1,024 KB.

For multi-stream tests, the skip parameter was used to position the reads or writes to start at different offsets of the device. For example, we used the following commands for the four-stream read test:

- `lmdd of=internal skip=0 if=/dev/raw/raw10 bs=32k count=8192`
- `lmdd of=internal skip=8192 if=/dev/raw/raw10 bs=32k count=8192`
- `lmdd of=internal skip=16384 if=/dev/raw/raw10 bs=32k count=8192`
- `lmdd of=internal skip=24576 if=/dev/raw/raw10 bs=32k count=8192`

For multiple-stream I/O tests, the aggregate performance (MB/s) was calculated as the total number of bytes divided by the longest elapsed time.

All tests were run on a quiet system. There were no other activities on the CX600 when the tests were running.

B.1.4 Read Performance Results

Figure B-1 and Table B-1 show the read performance of the Qlogic 2340 HBA with different I/O sizes and numbers of processes. The results show the performance of reading 1 GB of data from the CX600 storage array. Each test was run twice consecutively, and the first read test was to fill up the controller cache for the subsequent read. The measurements were obtained from the second reads. Since the read cache on the CX600 SPE controller was 1,400 MB, all the second reads would have been reading directly from the controller cache.

[5] The complete Lmbench benchmark kit can be found at http://www.bitmover.com/lmbench/.

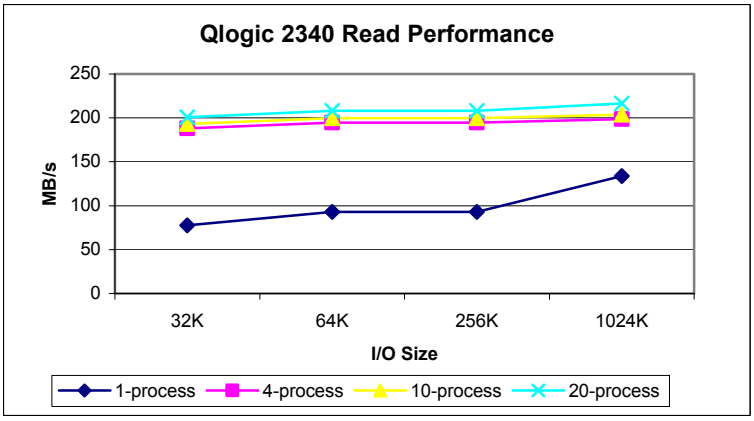

Figure B-1. Read performance of the Qlogic 2340 HBA.

Table B-1. Qlogic 2340 Read Performance (MB/s).

	32 K	64 K	256 K	1,024 K
1-process	77.47	92.95	92.95	133.74
4-process	188.10	194.57	194.57	198.66
10-process	193.08	199.86	199.86	203.66
20-process	200.65	208.01	208.01	216.33

These results seem to indicate that, except for the single-stream read tests, the 2340 HBA was able to achieve a read performance close to 200 MB/s with multiple streams and an I/O size larger than 64 KB. The best performance was achieved with the 20-process test using an I/O size of 1,024 KB.

B.1.5 Write Performance Results

Figure B-2 and Table B-2 show the write performance of the Qlogic 2340 HBA with different I/O sizes and numbers of processes. The results show the performance of writing 1 GB of data to the CX600 storage array. Each test was run twice consecutively, and the measurements were obtained from the first writes. Since the write cache on the CS 600 SPE controller was 2,000 MB, the reads would have been writing directly to the controller cache.

These results seem to indicate that the 2340 HBA was able to achieve a write performance that is close to 200 MB/s. The best write performance was achieved with the 20-process test and an I/O size of 1,024 KB.

It is not clear why we were not able to see more test results that are close to 200 MB/s. This lower write performance for some cases could have been caused by the write-behind activity on the CX600 to flush data that were written from the earlier write tests. We had the CX600 for testing for only a short period and we did not have enough time to re-run the tests to investigate these write anomalies.

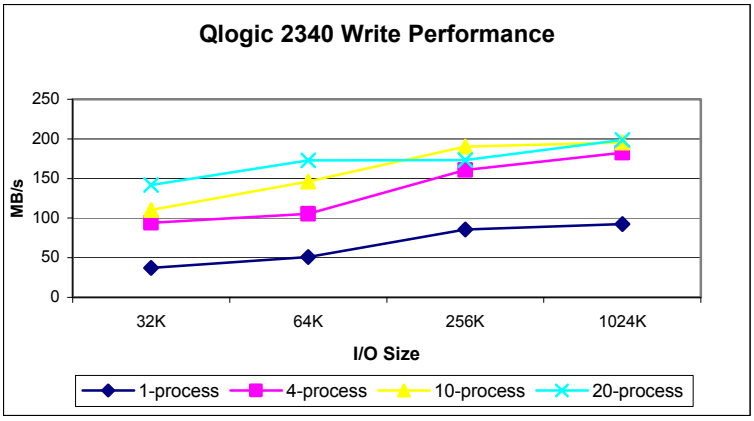

Figure B-2. Write performance of the Qlogic 2340 HBA.

Table B-2. Write Performance (MB/s) of the Qlogic 2340 HBA.

	32 K	64 K	256 K	1,024 K
1-process	37.29	50.97	85.58	92.44
4-process	94.28	105.50	160.84	182.60
10-process	110.29	146.33	190.38	195.28
20-process	141.81	172.83	173.14	198.73

The Qlogic 2340 was capable of sustaining the read and write performance at 200 MB/s. The Qlogic 2340 HBA was selected for use in all the new compute nodes of the expanded GUPFS testbed.

B.2 GFS 4.2 Performance

This section describes the GFS 4.2 performance.

B.2.1 Linux Host Configuration

The test host is the GUPFS cluster, a Linux platform with six nodes. All nodes have same basic hardware configuration:

- Intel Server Board STL2 motherboards
- dual 1 GHz P3 processors
- 1 GB 133 MHz ECC memory
- 18 GB 10,000 RPM LVD SCSI U160 disk drives
- one IDE CD-ROM
- one floppy drive
- RedHat 7.2 with GFS-modified 2.4.9 kernel
- one Qlogic QLA2200 1 Gb HBA
- modified Qlogic v4.27 driver that supports the DMEP protocol.

B.2.2 Storage and Pools

The storage used for the GFS performance test is a DotHill SANnet 7124 storage system with the following configuration:

- two disk trays, each with 10 10K RPM 73 GB disks
- two controllers, each with two 1 Gb/s host ports, one primary and one secondary (the test used the two primary ports of both controllers)
- 1 GB cache per controller, with read and write caching enabled
- two Raid 5 LUNs, each with six disks from one tray, were configured for the test
- hosts and DotHill were connected to a Brocade SilkWorm 2800 (1 Gb/s) switch.

There were two pool devices created for the GFS file system: **`dh_dmep0_cidev`** and **`dh_dmep0`**. The first one, **`dh_dmep0_cidev`**, was a GFS cluster information device (**`cidev`**) and was created on the first cylinder of the first Raid 5 LUN. The second pool device **`dh_dmep0`**, where the GFS file system was made, was a two-way stripe on the two Raid 5 LUNs, using a stripe size of 128 sectors (64 KB).

The tests were performed using the Pioraw program as an MPI setup. Pioraw was modified to perform reading and writing on raw devices for performance comparison.

B.2.3 Complete GFS 4.2 Performance Results

Table B-3 provides a complete set of Pioraw results on the following six test configurations:

- /dev/sdb — one of the two raw devices used in the pool definition
- pool (lo) — a pool device used for the GFS file system; the host had 512 MB memory
- pool (hi) — the same pool device, except the test was run on a host with 1 GB memory
- gfs (dmep) — GFS 4.2 using hardware DEMP for locking; the host had 512 MB memory
- gfs (tcp) — GFS 4.2 with tcp lock daemon; the host had 512 MB memory
- gfs (tcp-hi) — GFS 4.2 with tcp lock daemon; the host had 1 GB memory

Table B-3. Complete GFS 4.2 Performance Results (MB/s).

Number of PEs: 1
Blocksize: 1024 bytes
WRITE

#blks	/dev/sdb	pool(lo)	pool(hi)	gfs(dmep)	gfs(tcp)	gfs(tcp-hi)
1	1.71	1.69	1.70	55.72	55.73	53.84
4	5.60	5.61	5.59	95.08	95.20	98.51
16	12.95	13.26	13.26	119.89	119.10	130.40
64	21.93	21.28	21.26	113.88	117.92	129.82
256	29.23	29.68	29.48	106.24	105.96	123.66
1024	33.39	35.70	35.32	115.69	109.17	127.68
4096	34.25	35.52	35.33	110.00	111.56	130.47
16384	34.24	35.66	35.37	92.38	93.04	132.94

[0] Filesize: 16777216 bytes

READ

#blks	/dev/sdb	pool(lo)	pool(hi)	gfs(dmep)	gfs(tcp)	gfs(tcp-hi)
1	3.39	3.32	3.22	234.82	233.50	237.32
4	9.62	9.32	8.51	235.08	234.65	235.77
16	30.54	29.43	22.63	247.31	244.24	242.46
64	47.01	43.84	34.11	249.87	248.03	251.99
256	53.83	25.52	34.90	161.98	159.53	170.10
1024	62.80	29.43	34.46	159.39	150.32	162.77
4096	62.73	29.38	34.80	150.02	153.01	163.04
16384	62.65	29.35	34.55	149.91	149.84	163.12

[0] Filesize: 16777216 bytes

Number of PEs: 2
Blocksize: 1024 bytes
WRITE

#blks	/dev/sdb	pool(lo)	pool(hi)	gfs(dmep)	gfs(tcp)	gfs(tcp-hi)
1	2.87	2.84	2.84	10.73	20.71	21.69
4	9.80	9.47	9.37	55.95	29.80	24.32
16	23.60	23.85	23.97	62.62	39.37	26.60
64	36.80	33.31	33.30	58.92	46.06	38.92
256	51.30	47.50	47.24	66.89	48.17	42.61
1024	57.01	55.98	56.90	57.13	52.85	48.52
4096	57.69	56.04	56.66	59.56	53.68	53.88
16384	56.39	56.52	56.92	50.46	55.81	62.02

[0] Filesize: 16777216 bytes

READ

#blks	/dev/sdb	pool(lo)	pool(hi)	gfs(dmep)	gfs(tcp)	gfs(tcp-hi)
1	5.04	5.08	5.08	467.68	470.91	477.76
4	15.61	15.31	14.49	469.86	469.73	470.37
16	43.70	46.27	38.31	491.13	489.88	485.94
64	62.77	60.54	51.51	498.43	498.46	505.87
256	58.89	38.71	59.79	315.14	317.69	348.68
1024	70.31	56.62	56.65	297.80	303.80	334.04
4096	70.16	56.57	56.47	298.10	302.93	334.50
16384	70.07	56.66	56.64	301.48	301.45	334.71

[0] Filesize: 16777216 bytes

Table B-3 (continued)

Number of PEs: 3
Blocksize: 1024 bytes
WRITE

#blks	/dev/sdb	pool(lo)	pool(hi)	gfs(dmep)	gfs(tcp)	gfs(tcp-hi)
1	3.69	3.64	3.64	4.50	14.17	13.12
4	12.81	12.75	12.73	26.56	28.15	25.67
16	28.95	29.34	29.27	33.81	32.31	33.40
64	46.26	47.11	47.26	32.07	35.65	36.21
256	62.79	64.84	64.67	39.33	40.49	39.71
1024	65.35	75.41	75.96	37.14	46.96	47.76
4096	65.31	75.52	76.39	34.18	49.27	54.44
16384	65.19	76.64	76.67	43.70	56.25	60.70

[0] Filesize: 16777216 bytes

READ

#blks	/dev/sdb	pool(lo)	pool(hi)	gfs(dmep)	gfs(tcp)	gfs(tcp-hi)
1	5.99	5.86	5.98	696.42	705.43	710.44
4	19.12	19.06	20.23	700.14	705.02	706.11
16	58.24	67.78	55.67	730.48	736.81	732.59
64	77.25	93.84	83.52	743.82	749.68	756.43
256	73.26	65.74	73.06	471.44	476.48	520.31
1024	82.52	83.40	68.45	450.12	452.91	497.85
4096	82.46	83.47	68.54	447.92	448.72	498.60
16384	82.48	83.47	68.58	448.59	450.12	498.71

[0] Filesize: 16777216 bytes

Number of PEs: 4
Blocksize: 1024 bytes
WRITE

#blks	/dev/sdb	pool(lo)	pool(hi)	gfs(dmep)	gfs(tcp)	gfs(tcp-hi)
1	4.02	4.24	4.24	2.65	15.25	15.44
4	14.15	15.09	15.10	23.92	29.37	31.70
16	27.59	35.83	35.58	23.74	35.01	34.68
64	48.10	59.78	60.28	30.11	36.68	37.95
256	66.86	77.83	77.50	29.86	40.13	40.93
1024	69.28	83.81	82.81	32.42	44.94	45.46
4096	69.23	83.89	82.83	38.15	47.96	51.00
16384	69.20	82.98	83.04	44.99	54.98	60.46

[0] Filesize: 16777216 bytes

READ

#blks	/dev/sdb	pool(lo)	pool(hi)	gfs(dmep)	gfs(tcp)	gfs(tcp-hi)
1	6.35	6.34	6.44	928.50	943.47	942.16
4	22.94	23.37	22.89	932.13	940.96	940.18
16	69.24	83.37	71.14	981.06	982.64	969.40
64	89.47	122.64	109.07	991.90	979.46	1006.85
256	95.92	114.13	102.97	636.45	637.09	692.96
1024	96.53	105.81	101.17	610.11	612.93	662.30
4096	96.54	106.28	102.64	598.13	607.31	663.23
16384	96.54	106.18	102.90	600.01	600.21	663.52

[0] Filesize: 16777216 bytes

Table B-3 (continued)

```
Number of PEs: 5
Blocksize: 1024 bytes
WRITE
#blks  /dev/sdb  pool(lo)  pool(hi)  gfs(dmep)  gfs(tcp)  gfs(tcp-hi)
    1      3.03      4.80      4.78       7.67     12.35        14.96
    4     11.74     16.99     16.96      19.06     24.90        34.77
   16     27.21     43.11     42.87      28.22     30.07        39.00
   64     52.72     66.51     65.96      21.38     33.43        41.32
  256     72.03     84.79     85.00      28.37     37.91        41.12
 1024     72.46     87.12     87.08      33.23     41.82        44.22
 4096     72.49     87.14     87.07      34.33     44.54        50.28
16384     72.51     87.09     87.07      41.80     53.22        56.96
[0] Filesize: 16777216 bytes

READ
#blks  /dev/sdb  pool(lo)  pool(hi)  gfs(dmep)  gfs(tcp)  gfs(tcp-hi)
    1      5.00      8.25     10.65    1171.93   1175.91      1178.13
    4     18.89     24.13     22.63    1171.19   1175.04      1174.38
   16     75.96     97.15     83.72    1223.62   1247.75      1225.31
   64     90.91    140.12    144.24    1244.21   1220.25      1255.47
  256     77.12    102.77    132.48     789.06    790.86       872.38
 1024     92.97    115.96    139.17     751.42    752.89       835.08
 4096     92.92    115.65    133.97     754.71    754.84       836.27
16384     92.80    115.37    142.06     749.14    751.83       836.56
[0] Filesize: 16777216 bytes
```

B.3 The Pool Cluster Volume Manager

The Pool Volume Manager is a simple logical volume manager that came with GFS 4.2 to allow multiple raw devices combined into one greater device through striping and concatenation. In this test, we were interested in comparing the I/O performance on the Pool Volume Manager.

B.3.1 Test Setup

The test host is one of the old GUS nodes with the Intel STL2 motherboard configuration. The storage used for the test is the DotHill array. The **`lmdd`** program was used. For multiple-stream I/O tests, the aggregate performance (MB/s) was calculated as the total number of bytes divided by the longest elapsed time. All tests were run on a quiet system.

B.3.2 Pool Device Configuration

The pool created for the tests is a two-way stripe using two Raid-5 LUNs on the DotHill storage array. Table B-4 shows the output from the **`pinfo`** command.

Table B-4. GFS 4.2 Pool Information.

```
*********************************************************
* Pool information:                                     *
* Pool name           :                      dh_dmep0   *
* Number of subpools  :                             1   *
* Total Capacity      :                      683.1 GB   *
* In use              :                            NO   *
*=======================================================*
* Subpool 0                                             *
*   Number of devices :                             2   *
*   Stripe size       :                           128   *
*   Total weight      :                             2   *
*   Type              :                      gfs_data   *
* - - - - - - - - - - - - - - - - - - - - - - - - - - - -*
*     Device                /dev/sdb2                   *
*       Weight          :                             1   *
*       Blocks          :                     716322176   *
* - - - - - - - - - - - - - - - - - - - - - - - - - - - -*
*     Device                /dev/sdc2                   *
*       Weight          :                             1   *
*       Blocks          :                     716322176   *
*********************************************************
```

B.3.3 Pool Performance

Figure B-3 shows the I/O performance of a two-way striped pool and that of a single raw device participating in the stripe. Since the pool was striped over two devices, theoretically the pool should be able to sustain twice the I/O rate that a single device can sustain.

The tests were run with 128 MB and 1 GB file sizes using different I/O sizes. The charts show only the results using the 1,024 KB I/O size.

Since the test host was running the Linux 2.4.9-gfssmp kernel with 1 GB system memory, there was a potential of performance degradation caused by the bounce buffer problem. The tests were also run with a lower system memory size (512 MB) to see how the bounce buffer problem may affect the performance. The results were shown as **`pool(lo)`** and **`raw(lo)`**.

The read performance on a single pool device was very poor, comparing to what the underlying device could do. This does not seem to be a bounce buffer issue since the poor read performance was also observed with the configuration with less system memory on the host. Also the pool device did not seem to scale well when the # of processes increased. The read performance on a single pool device remained at about 40MB/s, independent of how many read processes were using.

The bounce buffer problem did seem to affect the write performance, on both the raw devices and the pool device. We experienced the bounce buffer problem on the host with 1GB system memory. The write performance on both the pool device and the raw device remained at less than 50MB/s for writing a 128MB file and 40MB for wring a 1GB file. When we reduced the system memory to

512MB on the host, we saw much better write performance on the pool device and the raw device. It was a surprise to see the good scalability on writes on the pool device for the case of writing a 128MB file. For 16 processes, we were able to achieve a write rate at 150MB/s. The result shows a 25% overhead caused by the pool device layer.

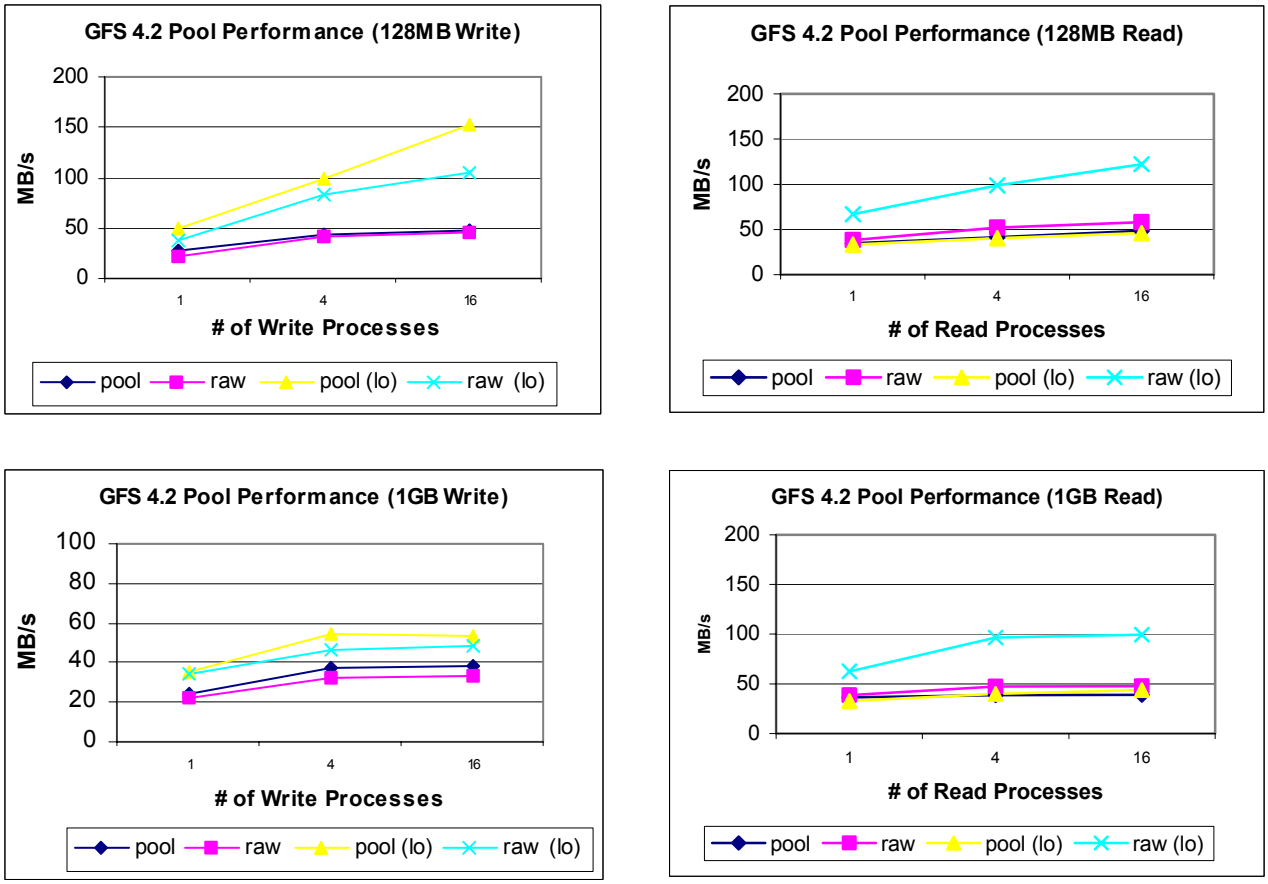

Figure B-3. GFS 4.2 Pool performance.

B.3.4 Complete Pool Device Performance Results

The complete read performance results are shown in Table B-5. The complete write performance results are shown in Table B-6.

Table B-5. GFS 4.2 Pool Device Read Performance (MB/s).

Device	File Size	N	32 K		64 K		256 K		1024 K	
			1st	2nd	1st	2nd	1st	2nd	1st	2nd
Pool (1 GB)	128 MB	1	28.62	31.60	33.98	37.95	35.25	36.47	31.59	34.76
		4	38.69	39.60	39.37	39.20	38.58	39.70	39.95	41.22
		16	39.28	41.45	43.13	43.10	44.32	42.59	43.66	48.45
	1 GB	1	28.65	30.60	34.19	38.80	35.09	39.27	34.21	36.48
		4	39.12	38.94	39.28	40.23	41.41	38.29	40.20	38.71
		16	37.63	38.59	40.59	39.21	41.24	39.14	41.20	39.16
Raw (1 GB)	128 MB	1	30.04	33.35	35.86	40.04	41.12	45.64	38.70	38.43
		4	44.39	49.80	42.27	42.12	48.18	52.11	47.94	52.01
		16	49.69	57.64	50.11	58.96	51.58	60.98	53.89	58.10
	1 GB	1	30.04	30.09	35.17	35.72	41.05	40.67	38.64	38.70
		4	45.21	45.43	42.18	42.05	47.80	48.13	47.32	47.42
		16	46.24	46.74	46.62	47.48	47.79	47.94	47.97	47.77
Pool (512 MB)	128 MB	1	37.33	43.37	42.64	50.60	25.52	32.31	29.24	33.24
		4	50.75	53.36	49.52	42.85	40.22	41.80	37.41	40.40
		16	49.84	51.70	47.01	46.80	44.49	48.88	45.29	46.29
	1 GB	1	37.92	41.73	42.98	48.97	25.45	31.74	29.14	32.96
		4	52.14	52.05	50.80	43.36	41.95	41.33	42.61	40.09
		16	48.27	47.15	45.34	44.43	44.06	46.34	43.86	43.98
Raw (512 MB)	128 MB	1	39.69	46.09	46.01	54.87	54.22	63.52	62.33	66.84
		4	61.22	76.07	63.36	95.73	94.77	98.10	97.30	98.79
		16	77.51	111.51	98.11	113.16	103.38	116.54	99.96	122.35
	1 GB	1	39.98	39.94	46.81	46.74	54.77	55.26	62.55	62.58
		4	62.07	62.11	75.26	75.89	94.17	94.82	95.90	96.34
		16	84.20	87.45	95.08	96.92	96.71	98.21	97.67	99.26

Table B-6. GFS 4.2 Pool Device Write Performance (MB/s).

Device	File Size	N	32 K		64 K		256 K		1024 K	
			1st	2nd	1st	2nd	1st	2nd	1st	2nd
Pool (1 GB)	128 MB	1	17.31	18.7	21.29	23.75	26.29	27.82	27.93	28.63
		4	36.78	40.24	39.3	40.97	43.41	43.6	43.42	43.74
		16	45.48	45.47	46.54	47.26	45.18	47.63	48.95	46.59
	1 GB	1	15.9	15.41	19.03	18.43	23.19	22.43	24.62	23.75
		4	30.12	28.52	32.48	31.3	39.9	39.12	39.1	36.98
		16	38.22	36.91	39.81	38.71	40.1	39.08	38.89	37.88
Raw (1 GB)	128 MB	1	16.87	15.09	20.83	18.28	25.26	22.17	26.73	22.67
		4	30.63	30.45	37.89	35.78	41.76	40.40	43.28	40.63
		16	46.90	45.60	46.61	46.24	48.96	47.00	47.58	45.60
	1 GB	1	15.15	14.91	18.50	18.15	22.48	22.18	22.99	22.71
		4	24.95	24.39	30.46	29.58	32.14	31.16	33.69	32.71
		16	35.06	34.02	33.39	32.43	34.38	33.14	34.17	33.23
Pool (512 MB)	128 MB	1	20.19	22.81	24.43	27.99	35.79	40.03	44.10	48.73
		4	45.74	52.14	58.52	65.61	82.49	92.71	91.38	99.02
		16	82.84	95.23	104.21	113.23	108.00	119.78	125.82	152.75
	1 GB	1	18.25	17.80	21.93	21.44	30.69	29.85	36.94	35.62
		4	35.32	33.14	43.41	39.86	56.28	52.14	60.38	54.87
		16	52.22	47.77	58.81	53.14	59.64	54.27	62.92	53.37
Raw (512 MB)	128 MB	1	21.18	18.69	25.76	22.62	38.31	31.47	44.73	36.64
		4	38.82	38.40	53.50	54.05	71.37	69.42	71.33	82.65
		16	72.93	80.41	77.40	92.40	81.87	100.81	84.65	104.24
	1 GB	1	18.41	18.02	22.04	21.65	30.93	30.18	36.70	34.50
		4	30.81	29.90	39.65	38.24	48.84	46.76	48.41	46.43
		16	45.43	43.50	48.99	46.93	51.20	49.00	50.98	48.70

B.4 EMC CX600 Performance Evaluation

Due to a short evaluation period, the focus of this test was to measure read and write performance with a single port on the EMC Clarion CX600. The tests were done with different file sizes—1 GB, 2 GB, 4 GB, and 8 GB—to see how data caching, read-ahead, and write-back implementation may affect the I/O performance.

The primary focus of this performance test was to measure read and write performance from a single storage port using a Linux host and a Qlogic 2340 HBA. The evaluation used the same test configuration we used earlier for the HBA performance test. The tests included single-stream and multi-stream tests with different I/O sizes.

B.4.1 CX600 Single-Port Read Performance

Figure B-4 shows the read performance of different file sizes using 1,024 KB as the I/O size.

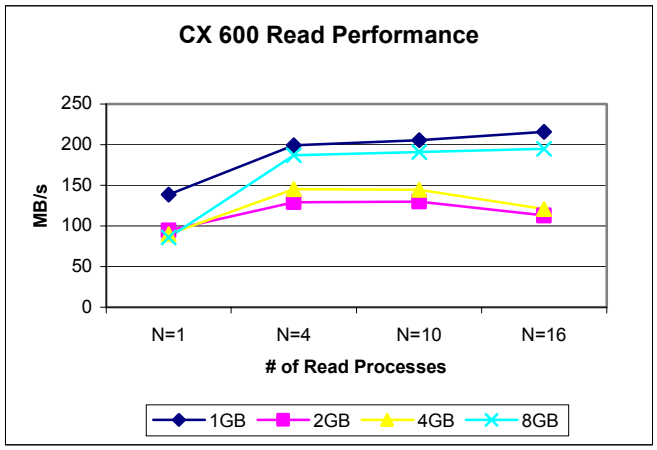

Figure B-4. EMC Clarion CX600 single-port read performance.

Each test was run twice to study how caching on the CX600 may affect the read performance. Since the read cache on each SPE controller was 1,470 MB, after the first read, the entire 1 GB file would have been in the controller cache. As expected, the first result was not much different from the second result for all test cases, except for the one-process 32 K read test.

The result also shows the best possible read performance between the host and CX600 for different test cases. The port performance for reads is about 200+ MB/s.

The performance results of the 2 GB, 4 GB, and 8 GB read tests are shown in Table B-7. For single-stream reads, the best performance is close to 90 MB/s. For multi-stream reads, the best performance is close to 200 MB/s (195.70 MB/s for the 20-process, 1,024 K, 8 GB test).

Table B-7. EMC Clarion CX600 Single-Port Read Performance (MB/s).

File Size	N	32 K		64 K		256 K		1,024 K	
		1st	2nd	1st	2nd	1st	2nd	1st	2nd
1 GB	1	66.32	77.47	91.11	92.95	126.74	125.67	138.56	133.74
	4	188.19	188.10	195.14	194.57	198.43	198.16	199.05	198.66
	10	193.33	193.08	199.58	199.86	203.55	204.98	205.52	203.66
	20	200.20	200.65	208.36	208.01	214.83	215.18	216.66	216.33
2 GB	1	65.80	51.87	73.91	68.76	81.67	73.55	94.61	82.95
	4	81.03	58.29	103.58	56.24	105.04	55.31	129.05	78.11
	10	85.46	62.31	111.85	55.20	104.11	62.37	130.22	77.62
	20	79.37	64.50	110.51	66.06	102.50	58.63	113.37	62.37
4 GB	1	59.81	58.25	71.18	68.49	77.67	74.88	89.86	80.94
	4	82.94	81.42	109.95	107.47	105.01	107.26	145.10	142.29
	10	109.78	72.57	130.75	89.24	122.33	93.49	144.39	110.99
	20	134.76	75.53	141.77	95.63	129.19	95.32	120.61	103.02
8 GB	1	56.51	57.48	70.66	68.85	75.31	76.97	86.10	85.35
	4	162.02	162.48	190.98	190.45	194.88	195.58	186.92	195.55
	20	157.20	137.06	177.03	165.83	186.46	174.21	190.98	177.36
	16	152.51	147.01	176.84	157.30	167.85	151.53	194.79	195.70

The 2 GB read performances are somewhat disappointing. These results may have something to do with how much data was cached in the controller cache. However, it is not clear why the 4 GB and 8 GB results are generally better than 2 GB results. Further investigation is needed.

These results show that we were able to achieve close to 200 MB/s performance with larger blocks and multiple streams.

B.4.2 CX600 Single-Port Write Performance

Figure B-5 shows the write performance on a single-port on CX600 using an I/O size of 1,204 KB with different file sizes.

For small files, we expected to see good write performance, when the entire file can completely fit in the write cache (2,000 MB) on the CX600 controller. However, the CX600 controller seemed to perform very well even with larger files (larger than the write cache size). This seems to indicate that the CX600 controller is capable of flushing a large amount of data to the disks faster than a single host could write to the controller.

For single-stream writes, the best write performance was about 90 MB/s. For multi-stream writes, the best performance was close to 200 MB/s when there were enough write streams.

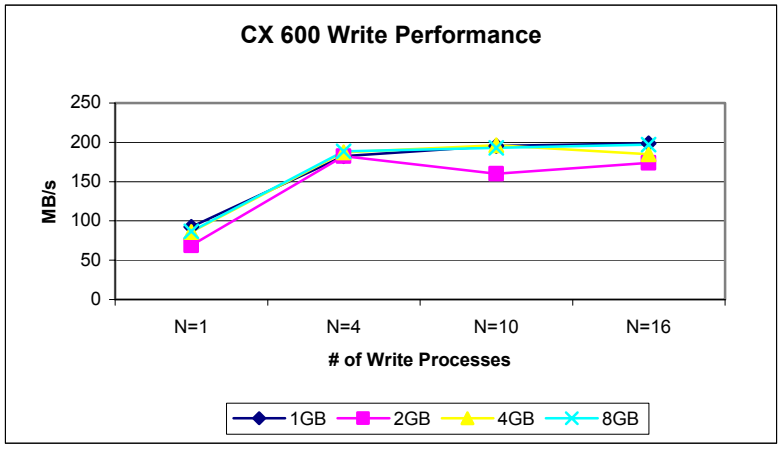

Figure B-5. EMC Clarion CX600 single-port write performance.

Table B-8 shows the complete single-port performance for different file sizes and different I/O sizes. Each test was run twice to study how caching and write-back implementation on the CX600 SPE controller may affect the write performance. It is not clear why in some cases the second write was much slower than the first write. Further investigation of the cache statistics on the CX600 may be needed to solve this mystery.

Table B-8. EMC Clarion CX600 Single-Port Write Performance (MB/s).

File Size	N	32 K		64 K		256 K		1,024 K	
		1st	2nd	1st	2nd	1st	2nd	1st	2nd
1 GB	1	37.29	34.99	50.97	52.25	85.58	76.10	92.44	86.44
	4	94.28	93.38	105.50	104.86	160.84	116.24	182.60	115.31
	10	110.29	113.31	146.33	95.71	190.38	94.04	195.28	82.92
	20	141.81	116.41	172.83	85.08	173.14	161.87	198.73	185.09
2 GB	1	36.03	35.59	45.26	51.99	63.42	78.31	68.63	88.23
	4	95.35	90.07	101.84	107.25	163.40	138.80	182.35	151.96
	10	114.10	117.04	146.91	120.92	161.21	150.64	159.92	152.04
	20	124.10	124.88	136.12	131.26	156.91	149.28	173.76	145.53
4 GB	1	35.28	36.95	50.46	50.80	71.19	79.87	85.72	89.51
	4	105.44	103.92	128.03	126.86	170.13	169.91	187.32	188.27
	10	120.01	119.39	153.11	151.63	194.79	168.24	196.45	170.11
	20	144.78	138.78	170.49	163.80	178.57	174.88	184.65	198.45
8 GB	1	35.38	36.48	51.00	51.28	76.27	78.04	86.70	90.72
	4	115.53	114.67	140.42	141.60	179.01	179.22	188.41	186.68
	20	140.91	138.93	163.81	162.24	192.34	193.02	193.14	193.18
	16	151.95	151.14	177.07	176.88	195.00	196.19	196.92	197.57

B.5 Yotta Yotta Evaluation

A major objective of the Yotta Yotta evaluation is to study whether the NetStorager GSX 2400 was able to break the 1 GB/s barrier and how GSX 2400 scales when the number of clients increases.

B.5.1 Linux Host Configuration

The test host is a Linux platform with the following configuration:

- SuperMicro P4DP6 motherboard, Intel E7500 chip
  - two Intel P4 2.2 GHz Xeon processors
  - 2 GB DDR PC2100 ECC/Reg. RAM
  - two 64-bit 100 MHz PCI-X
  - four 64-bit 66 MHz PCI
- RedHat 7.3 with 2.4.18-5 kernel
- One Qlogic QLA2340 2Gb HBA sitting on a PCI-X slot with a modified Qlogic v6.1b2 driver with the bounce buffer patch.

B.5.2 Yotta Yotta NetStorager GSX 2400 Configuration

The NetStorager GSX 2400 had the following configuration:

- disk capacity: 10K RPM, 73 GB
- four control blades, each with four 2 Gb host ports (only one port was used)
- 4 GB cache per blade; read and write caching was enabled
- a single seven-way RAID 0+1 LUN was configured for the test
- disks in the same mirrored pair were placed on separate disk enclosures
- GSX 2400 was attached to a Qlogic SANbox2 switch.

B.5.3 GSX 2400 Single-Port Read Performance

Figure B-6 shows the single-port read performance with an I/O size of 1,024 KB.

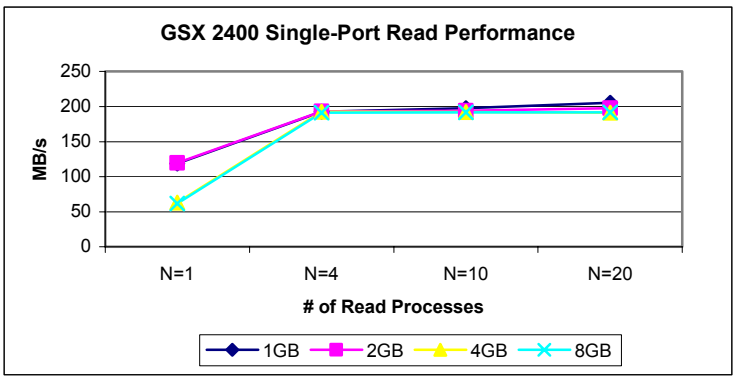

Figure B-6. Yotta Yotta NetStorager GSX 2400 single-port read performance.

Each I/O test was run twice to study how caching and the read-ahead implementation on GSX 2400 may affect the read performance. Only the results from the second reads are shown in the figure.

The cache on each blade is 4 GB, and about 3 GB is useable for user data. Therefore, for 1 GB and 2 GB files, after the first read, the entire files would have been cached in the blade cache.

The result shows the possible read performance on a single-port on GSX 2400 is about 200+ MB/s. It begins to saturate the storage device after four processes, independent of the file size.

The complete single-port read performance is shown in Table B-9.

Table B-9. Yotta Yotta NetStorager GSX 2400 Single-Port Read Performance (MB/s).

File Size	N	32 K		64 K		256 K		1,024 K	
		1st	2nd	1st	2nd	1st	2nd	1st	2nd
1 GB	1	24.29	52.75	73.31	73.27	107.42	107.48	116.14	119.02
	4	161.58	162.50	172.19	172.11	194.47	194.28	192.60	192.85
	10	164.54	164.59	175.54	174.72	194.80	195.58	198.10	197.49
	20	148.84	148.37	185.16	185.36	200.99	202.70	205.70	205.37
2 GB	1	30.30	54.39	75.51	75.08	108.46	108.48	120.33	119.75
	4	162.51	162.46	170.93	170.91	193.38	193.54	192.42	193.25
	10	163.64	163.80	171.04	173.29	194.96	191.38	194.13	194.10
	20	142.55	143.17	180.52	180.98	195.86	197.79	196.31	197.63
4 GB	1	31.17	49.49	49.36	63.26	69.28	46.84	86.39	63.19
	4	93.93	92.75	114.00	125.17	144.61	182.27	135.89	192.69
	10	137.16	163.34	153.18	170.72	185.12	191.62	185.57	192.37
	20	143.19	141.42	178.90	179.34	194.37	194.14	194.58	191.27
8 GB	1	26.77	39.30	42.23	49.85	55.63	47.22	73.04	61.87
	4	113.20	158.12	143.84	133.47	158.44	183.10	163.92	191.38
	20	151.43	162.50	158.41	171.12	186.74	192.90	188.61	191.51
	16	143.28	140.74	178.14	176.81	191.46	191.28	191.20	191.66

B.5.4 GSX 2400 Single-Port Write Performance

Figure B-7 shows the single-port write performance using an I/O size of 1,024 KB. Each I/O test was run twice to study how caching and the write-behind implementation on GSX 2400 may affect the write performance. For single-stream writes, the best performance, independent of the file size, is at about 67 MB/s. The write performance is around 70–90 MB/s, except for the case of four-process writing an 8 GB file.

The complete single-port write performance results is shown in Table B-10. In many cases, the four-process tests outperformed other tests with 10 or 20 processes, independent of the file size or block size used in the tests. We have no explanation for this anomaly. Further investigation is needed.

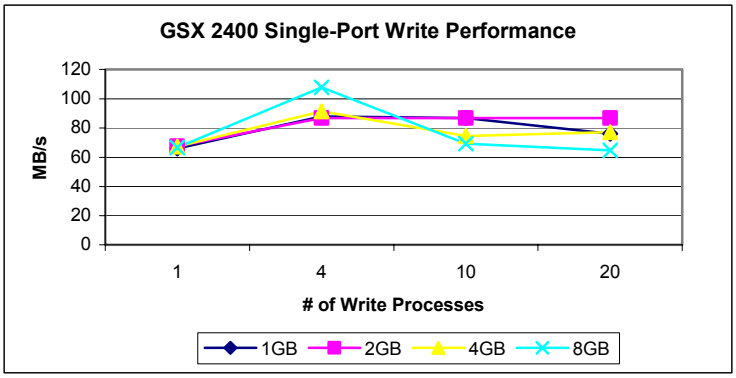

Figure B-7. Yotta Yotta NetStorager GSX 2400 single-port write performance.

Table B-10. Yotta Yotta NetStorager GSX 2400 Single-Port Write Performance (MB/s).

File Size	N	32 K		64 K		256 K		1,024 K	
		1st	2nd	1st	2nd	1st	2nd	1st	2nd
1 GB	1	36.14	36.03	28.61	48.55	33.62	64.59	39.74	66.12
	4	66.72	63.96	82.41	74.08	103.36	101.34	87.07	87.79
	10	69.63	59.25	102.97	99.86	101.78	109.72	86.08	86.88
	20	164.52	89.35	181.38	67.86	64.86	68.74	75.75	75.96
2 GB	1	35.39	36.06	27.59	49.16	32.53	64.64	33.58	67.46
	4	65.27	62.74	76.30	73.22	105.17	100.44	85.87	86.80
	10	82.55	81.41	90.94	95.13	98.78	108.92	85.93	86.72
	20	117.14	87.43	183.11	63.48	101.69	68.42	87.53	86.86
4 GB	1	33.37	35.45	46.07	48.53	63.99	64.14	67.01	67.25
	4	77.20	76.62	95.94	96.00	75.68	81.63	108.94	91.32
	10	62.59	65.98	71.16	77.14	79.00	78.00	73.38	74.63
	20	75.86	78.95	80.89	77.88	82.06	95.61	70.55	77.20
8 GB	1	35.34	35.59	47.84	48.59	63.86	63.24	66.43	66.63
	4	79.48	80.59	103.06	101.83	107.25	107.45	112.25	107.82
	20	63.39	69.81	57.86	56.55	64.69	61.46	76.12	69.19
	16	47.48	55.25	51.66	62.47	61.70	61.69	69.77	64.79

B.5.5 GSX 2400 I/O Performance Scalability

One very important performance measurement for evaluating the storage subsystem is the I/O performance scalability. A simple measurement is to determine the aggregate read/write performance on a single, shared LUN when the number of host connections increases.

A major objective for the GUPFS project's FY 2003 plan is to demonstrate/test 1 GB/s sustained file system performance. In order for a file system to sustain the 1 GB/s goal, the underlying storage must be capable of sustaining more than 1 GB/s. For a shared-disk file system, all hosts

may need to access the same underlying device at the same time. This 1 GB/s goal has been extremely difficult to achieve when the I/Os are performed on a single device from multiple hosts. One of the Yotta Yotta evaluation goals is to study how the GSX 2400 scales and whether it is able to break the 1 GB/s barrier.

The GSX 2400 test unit has four blades with two ports on each blade. However, due to the stability of the test environment, we were only able to use up to seven hosts connected to the seven ports on the GSX 2400. The performance results show a very good scalability when the number of host connections increases. With seven host connections, we were able to achieve the 1 GB/s objective. This, we believe, is a major technology milestone that is important for the deployment of a shared-disk file system.

Figure B-8 and Table B-11 show how the GSX 2400 scales when the number of host connections increases.

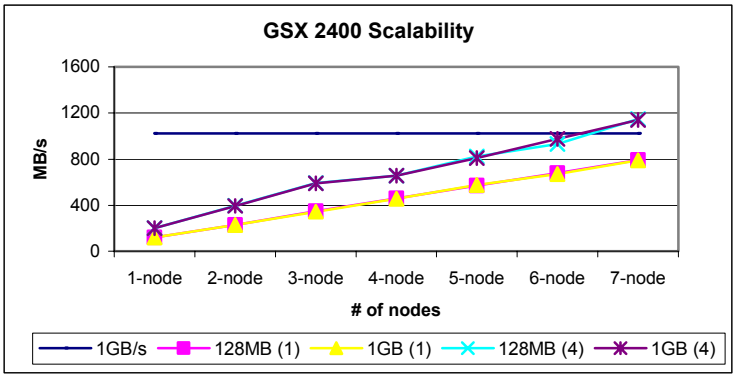

Figure B-8. Yotta Yotta NetStorager GSX 2400 scalability.

Table B-11. Yotta Yotta NetStorager GSX 2400 Aggregate Read Performance (MB/s).

N	File Size	1-node	2-node	3-node	4-node	5-node	6-node	7-node
1	128 MB	118.09	227.72	349.62	459.20	571.89	679.86	791.89
	1 GB	117.89	228.69	344.35	457.14	576.05	669.74	790.03
4	128 MB	198.34	396.69	594.36	657.18	823.90	932.13	1148.51
	1 GB	198.31	393.08	589.65	656.20	810.24	975.17	1138.67

Table B-12 shows the average per-node performance. Due to some stability issues, the test was set up so that in the first three test configurations (1-node, 2-node, and 3-node), each had one host connected to a port on a single blade, while in the last four tests (4-node, 5-node, 6-node, and 7-node), each had two hosts connected to a blade. It seems that when a blade had only one host connection, each blade was capable of doing 198 MB/s. When a blade had two host connections, the per-node rate dropped to around 160 MB/s (or 320 MB/s per blade). This 320 MB/s could be limited by the bus speed.

Table B-12. Yotta Yotta NetStorager GSX 2400 Average Per-Node Read Performance (MB/s).

N	File Size	1-node	2-node	3-node	4-node	5-node	6-node	7-node
1	128 MB	118.09	113.86	116.54	114.80	114.38	113.31	113.13
	1 GB	117.89	114.35	114.78	114.29	115.21	111.62	112.86
4	128 MB	198.34	198.35	198.12	164.30	164.78	155.36	164.07
	1 GB	198.31	196.54	196.55	164.05	162.05	162.53	162.67

Overall, we have been very happy with the GSX 2400's performance. Since the product is still in beta state, we have experienced many stability issues that have significant impact on our evaluation schedule. However, we have had excellent support from the manufacturer. We believe this evaluation so far has been very successful and has been beneficial to both Yotta Yotta and the HPC community.

Appendix C
Acronyms

ASCI	Advanced Simulation and Computing Program (formerly Accelerated Strategic Computing Initiative)
CIFS	Common Internet File System
CPU	central processing unit
CRC	cyclical redundancy check
CVFS	CentraVision File System (ADIC)
DAE	disk array enclosure
DMA	Direct Memory Access Protocol
DMAPI	Data Management Application Program Interface
DMEP	Device Memory Export Protocol
DOE	U.S. Department of Energy
ECC	error-correcting code
ERCAP	Energy Research Computing Allocation Process
FC	Fibre Channel
FCIP	Fibre Channel over IP
FY	fiscal year
GB	gigabyte (8 gigabits)
Gb	gigabit
GB/s	gigabyte per second (8 Gb/s)
Gb/s	gigabit per second
GFS	Global File System (Sistina)
GigE	Gigabit Ethernet
GNBD	global network block device
GPFS	General Parallel File System (IBM)
GULM	Global Universal Lock Manager (Sistina)
GUPFS	Global Unified Parallel File System
HBA	host bus adapter
HCA	host channel adapter
HDF	Hierarchical Data Format

HP	Hewlett-Packard
HPC	high performance computing
HPSS	High Performance Storage System
HSM	hierarchical storage management
HSSDC	high speed serial data connector
IB	InfiniBand
IBx	InfiniBand Expansion
I/O	input/output
IOPS	I/O operations per second
IP	Internet Protocol
iSCSI	Internet Small Computer System Interface
KVM	keyboard, video, mouse switch
LAN	local area network
LBNL	Lawrence Berkeley National Laboratory
LUN	logical unit number
LVM	logical volume management
MDS	metadata server
MHz	megahertz
MPI	Message Passing Interface
MPI-I/O	Message Passing Interface-I/O
MPP	massively parallel processing
NAL	network abstraction layer
NAS	network attached storage
NERSC	National Energy Research Scientific Computing Center
NIC	network interface card
NFS	Network File System
OLTP	online transaction processing
OS	operating system
OSF	Oakland Scientific Facility
OST	object storage target
PCI	peripheral component interconnect
PCI-X	peripheral component interconnect extension
PDSF	Parallel Distributed Systems Facility

PI	principal investigator
POSIX	Portable Operating System Interface
PXE	Pre-Execution Environment (Intel)
RAID	redundant arrays of independent disks
RDMA	Remote Direct Memory Access Protocol
RISC	reduced instruction set computing
SAN	storage area network
SCSI	small computer system interface
SE	storage engine
SGSFS	Scalable Global Secure File System
SP	storage processor (not to be confused with IBM SP supercomputer)
SPE	storage processor enclosure
SRP	SCSI RDMA Protocol
SRU	storage resource unit
TB	terabyte
TCP	Transmission Control Protocol
TLB	translation lookaside buffer
TOE	TCP Offload Engine
UDP	User Datagram Protocol
VFS	Virtual Filesystem Switch or Virtual File System
VEx	Virtual Ethernet Exchange
VFx	Virtual Fibre Channel Exchange
VGA	video graphics array
XDSM	Data Storage Management Application Program Interface

Appendix D
References

1. NERSC Strategic Proposal FY2002–FY2006, http://www.nersc.gov/aboutnersc/pubs/Strategic_Proposal_final.pdf.

2. "Comparing Storage Area Networks and Network Attached Storage." White paper, Brocade Communications Systems, Inc., 2001, http://www.brocade.com/san/white_papers/pdf/SANvsNASWPFINAL3_01_01.pdf.

3. A. Benner, *Fibre Channel: Gigabit Communications and I/O for Computer Networks*. Boston, McGraw Hill, 1996.

4. FC: Fibre Channel Protocol, ftp://ftp.t10.org/t10/drafts/fcp/fcp-r12.pdf.

5. Julian Satran, "iSCSI" (Internet Small Computer System Interface) IETF Standard, January 24, 2003, http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-20.pdf.

6. Raj Bhagwat, Murali Rajagopal, and Ralph Weber, "Fibre Channel Over TCP/IP (FCIP)," IETF Draft Standard, August 28, 2002, http://www.ietf.org/internet-drafts/draft-ietf-ips-fcovertcpip-12.pdf.

7. SRP: SCSI RDMA Protocol, ftp://ftp.t10.org/t10/drafts/srp/srp-r16a.pdf.

8. Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee, "Frangipanni: A Scalable Distributed File System," *Proceedings of the Sixteenth ACM Symposium on Operating System Principles,* pp. 224-237, October 1997.

9. Global File System (GFS), Sistina Software, Inc., http://www.sistina.com/downloads/datasheets/GFS_datasheet.pdf.

10. StorNext File System, Advanced Digital Information Corporation (ADIC), http://www.adic.com/ibeCCtpSctDspRte.jsp?minisite=10000&respid=22372&section=10121.

11. ASCI Path Forward, SGSFS, 2001, http://www.lustre.org/docs/SGSRFP.pdf.

12. "Lustre: A Scalable, High-Performance File System," Cluster File Systems, Inc., November 2002, http://www.lustre.org/docs/whitepaper.pdf.

13. Matthew O'Keefe, "Accelerating Technical Computing with Sistina Global File System-Based Storage Clusters," Sistina Software, Inc., November 2001, http://www.sistina.com/downloads/whitepapers/Tech_WP102A.pdf.

14. Kenneth W. Preslan, “Implementing Journaling in a Linux Shared Disk File System,” *Proceeding of the Eighth Goddard Conference on Mass Storage Systems and Technologies* (held jointly with the Seventeenth IEEE Symposium on Mass Storage Systems), NASA/CP-2000-209888, pp. 351-378, March 2000.

15. Kenneth W. Preslan, “A 64 Bit, Shared Disk File System for Linux,” *Proceedings of the Seventh Goddard Conference on Mass Storage Systems and Technologies* (held jointly with the Sixteenth IEEE Mass Storage Systems), March 1999.

16. Yotta Yotta NetStorager GSX 2400, Yotta Yotta, Inc., http://www.yottayotta.com/pages/products/overview.htm.

17. InfinIO Shared I/O System, InfiniCon Systems, Inc., http://www.infinicon.com/pdf/InfinIO_7000_Data_Sheet.pdf.

18. Peter J. Braam, Michael Callahan, and Phil Schwan, “The InterMezzo Filesystem,” presented at the O’Reilly Perl Conference 4.0, July 2000, http://www.inter-mezzo.org/docs/perlintermezzo.pdf.

19. Peter J. Braam, “File Systems for Clusters from a Protocol Perspective,” presented at the Second Extreme Linux Topics Workshop, Carnegie Mellon University, June 1999, http://www.cs.cmu.edu/afs/cs/project/coda/Web/docdir/extremelinux99.pdf.

20. Peter J. Braam, “The Coda Distributed File System,” *Linux Journal,* No. 50, June 1998.

21. M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, “Coda: a Highly Available File System for a Distributed Workstation Environment,” *IEEE Transactions on Computers*, Vol. 39, No. 4, April 1990.

22. “Brocade SAN Design Guide,” Brocade Communications Systems, Inc., 2002, http://www.brocade.com/san/pdf/53-0000231-05.pdf.

23. “Building Open SANs with Multi-Switch Fabrics,” Qlogic Corporation, 2001, http://www.qlogic.com/documents/datasheets/knowledge_data/whitepapers/building_open_sans.pdf.