I/O Performance Challenges at Leadership Scale

Samuel Lang, Philip Carns,
Robert Latham, and Robert Ross
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
{pcarns,slang,robl,rross}@mcs.anl.gov

Kevin Harms and William Allcock
Argonne Leadership Computing Facility
Argonne National Laboratory
Argonne, IL 60439
{allcock,harms}@alcf.anl.gov

ABSTRACT

Today's top high performance computing systems run applications with hundreds of thousands of processes, contain hundreds of storage nodes, and must meet massive I/O requirements for capacity and performance. These leadership-class systems face daunting challenges to deploying scalable I/O systems. In this paper we present a case study of the I/O challenges to performance and scalability on Intrepid, the IBM Blue Gene/P system at the Argonne Leadership Computing Facility. Listed in the top 5 fastest supercomputers of 2008, Intrepid runs computational science applications with intensive demands on the I/O system. We show that Intrepid's file and storage system sustain high performance under varying workloads as the applications scale with the number of processes.

1. INTRODUCTION

The demands of computational science have resulted in ever larger parallel computers, and storage systems for these computers have struggled to match the rates at which applications generate data. Computational science applications are increasingly I/O bound because of the disparity in the rate of improvement in computational capacity compared to storage throughput. Today's top supercomputers maintain a delicate balance between the aggressive I/O demands of applications and the large cost of storage hardware. The 557 TFlops Blue Gene/P system (known as *Intrepid*) at the Argonne Leadership Computing Facility (ALCF) demonstrates storage provisioning that allows I/O performance to scale to a large percentage of the machine. But appropriate storage partitioning is not sufficient in achieving high performance. The challenge falls to the storage software as well, which has had to adapt to extract the highest possible performance from increasingly complex storage hardware. High-level parallel I/O libraries [1, 2], I/O middleware, and parallel file systems require increasing specialization to maintain high performance for the demanding I/O workloads of computational science.

 SC09 November 14-20, 2009, Portland, Oregon, USA
978-1-60558-744-8/09/11

In this paper we evaluate the scalability of the parallel storage system on a leadership-class machine. We characterize the I/O architecture of Intrepid and measure the effectiveness of the I/O software to manage data for computational science at the largest scales. In section 2, we discuss the approaches that others have taken in assessing the capabilities of storage hardware and software, and we describe how we adopt the best practices from prior research in our evaluation. Section 3 describes the hardware and software architectures studied in this work. Section 4 presents our evaluation of the entire system, starting with the capabilities of the storage hardware and building up to an analysis of application workload performance.

2. BACKGROUND

Our research is motivated by the increasing gap between computation and I/O performance and by the challenges of I/O software and hardware to meet the I/O demands of computational science applications. We assess previous work in this area, focusing on studies that perform scalability measurements of parallel storage hardware and software systems in high performance computing. Scaling studies focus to varying degrees on the storage system components, the parallel file system, and the application workloads. We categorize these studies into three broad classes of work, and discuss the degree of scalability achieved during evaluation.

2.1 Parallel File System Evaluations

A few studies focus on a particular file system and the overall design choices made to achieve high performance. In these studies, performance results are a validation of the design, and scalability is demonstrated to the degree considered appropriate for the targeted systems and workloads. Schmuck and Haskin [3] give an overview of the GPFS architecture, a shared disk file system that provides access to shared files by using distributed locking mechanisms. They demonstrate scalability for both clients and storage servers for tens of nodes with large contiguous access patterns and discuss optimizations for shared file access and block allocation. Welch et al. [4] present the architecture of the Panasas parallel file system, also a shared disk file system. Their performance results demonstrate scalability for hundreds of clients and tens of storage nodes with access patterns that perform both contiguous and random I/O to individual files.

Weil et al. describe Ceph [5], a cluster file system that uses variable-sized objects to store file data and includes a novel technique for distributing objects over storage nodes. Their

evaluation includes a scalability study of up to 32 storage nodes, keeping the process count constant at 400. A popular file system within HPC, the Sun Lustre file system also uses objects to store file data, and a number of scalability studies of systems that use Lustre have been performed [6][7], showing performance with as many as 8,192 clients and 144 storage nodes [7]. Oldfield et al. [8] approach scalability

Oberg et al. [9] compare GPFS, Lustre, and PVFS and document the variability in out-of-the-box performance for those file systems. They perform runs with tens to hundreds of client processes, focusing on some of the administrative challenges with the current state of the art in parallel storage software and some of the intensive tuning required by administrators to achieve reasonable performance.

2.2 HPC System Scaling Studies

The second category of scalability studies characterize, the I/O behavior of a specific production system with leading-edge requirements (at the time of study) for I/O capability, performance, and scalability. These works focus on the I/O patterns of applications that are targeted for their systems, and so perform large scale runs with benchmarks that mimic their application's access patterns, using the parallel file system deployed at that site. The primary focus of these studies is scalability, as many of the systems are the largest in the world, with unprecedented node counts and independent storage components. Moreover, they often have significant capability requirements, where a large fraction of the machine's I/O resources may be used for a single job [10]. Most of these studies perform tests at scales with processor counts in the thousands. These studies show that even at these scales, significant challenges arise. Laros et al. [10] measure the I/O performance of Red Storm, a Cray XT system with 25,920 cores, running the Lustre file system. Their methodology first measures each component of the I/O path and then, using the client to storage node ratio, measures scalability of the system up to that ratio. Their results demonstrate performance degradation at larger runs where the storage devices were oversubscribed (more than one client per storage device).

Two recent studies measure the scalability of the Jaguar system at Oak Ridge National Laboratory, a Cray XT3/XT4 machine with more than 32,000 cores in 2008. Yu et al. [11] measure the scalability of each level in the storage heirarchy and perform scalability runs at 8,192 processes. Fahey et al. [12] perform scalability measurements of the same system but focus on the performance of runs with a constant file size. Both studies identify a number of key tuning parameters required to get significant performance improvement but also demonstrate significant challenges to scaling I/O at thousands of processes. For example, in both studies of the Jaguar system, the time to open a file begins to dominate at larger node counts. Both studies also demonstrate significant performance improvement during shared-file tests by determining the optimal number of I/O aggregators to perform I/O (referred to as *subsetting* in [12]), decreasing the effective scale that the file system must support.

2.3 Application I/O Studies

The last category of studies focuses primarily on application I/O workloads and the degree to which I/O intensive applications are able to maintain efficient I/O performance with increasing node counts and problem sizes. Kandaswamy et al. [13] perform an early study of a variety of applications and measure how performance improves for various optimizations on each application. This study includes runs with as many as 512 processes. More recently, Borrill et al. [14] demonstrate performance variations of a single application benchmark MADbench2 on a number different systems and characterize some of the architectural and software challenges that are impediments to scalability. Shan et al. [15] attempt to parameterize the I/O workloads of a selected set of HPC applications in order to predict the performance of applications on different systems and to unify the set of tests needed for system testing and procurement. They focus on the use of the IOR synthetic benchmark to mimic application I/O behavior. This study performs runs ranging from 8 to 512 client processors. While each of these studies covers a wide variety of applications and their differing I/O workloads, testing is limited to hundreds of clients per run.

One of the few studies of an application running at a much larger scale was done by Fisher et al. [16] on the Blue Gene/L system at Lawrence Livermore National Laboratory, performing FLASH3 runs on 64,000 processors. While this was not an I/O study, they document some of the major challenges of I/O at this scale.

In our analysis we adopt the best characteristics of all three classes of studies. Like parallel file system studies, we highlight how specific features of the file system and I/O software help to attain high performance. Like HPC scaling studies, we perform a thorough analysis of the underlying hardware and test PVFS with synthetic benchmarks running on over 100,000 processes. Moreover, we use a set of application-oriented benchmarks to provide insight into how the file system will perform for computational science applications. Overall we present a unique and detailed picture of a computational science I/O system at extreme scale.

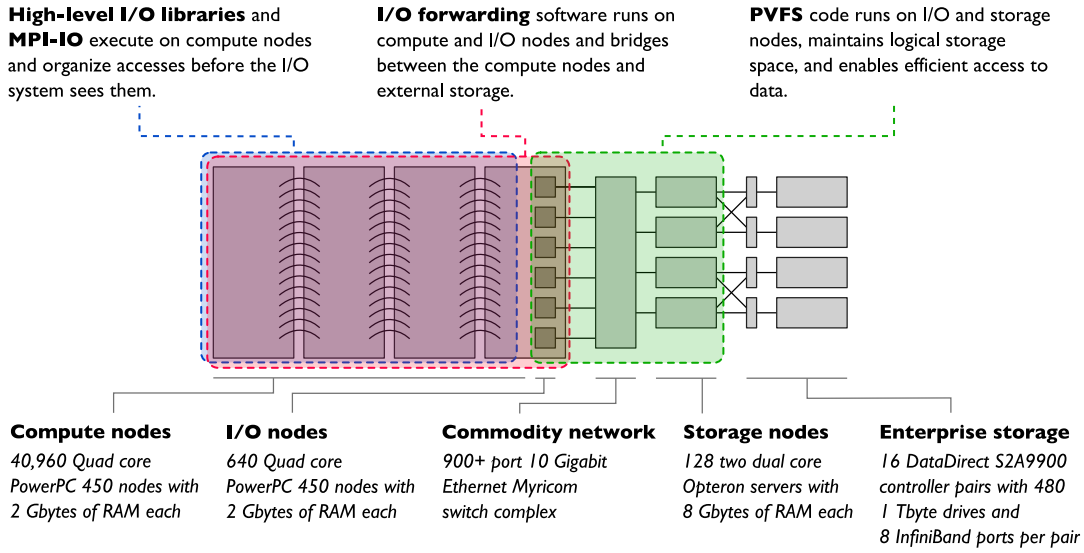

Figure 1: Architectural diagram of Intrepid.

3. SYSTEM ENVIRONMENT

Leadership-computing systems are on the cutting edge of computing hardware and system software, and technologies that first appear in these systems often "trickle down" into future systems. Many of the I/O hardware and software components have unique characteristics and features needed by leadership-computing systems that perform computational science. In this section we describe the architecture of Intrepid on which this evaluation is performed, present the I/O software deployed on this system, and focus on some of the important features needed by computational science applications running on Intrepid.

3.1 Intrepid I/O Architecture

The IBM Blue Gene/P (BG/P) system is the second in a series of supercomputers designed by IBM to provide extreme-scale performance along with high reliability and best-in-class power consumption. Figure 1 shows the architecture of Intrepid, the Argonne Leadership Computing Facility BG/P system, including the storage hardware attached externally to the machine. Each rack of BG/P has 1,024 quad-core nodes with a total of 2 terabytes of memory and a peak performance of 13.9 teraflops. Intrepid has 40 such racks; in aggregate, the system has 80 terabytes of memory, a peak performance of 557.06 TFlops, and 163,840 compute cores.

The BG/P compute node I/O and interprocess communication travel on separate internal networks. A three-dimensional torus network is used for communicating among compute nodes (abbreviated *CNs*), while a tree network allows CNs to perform I/O to designated I/O forwarding nodes. In the BG/P system, these I/O forwarding nodes are referred to simply as *I/O nodes* (abbreviated *IONs*), and are distinct from the storage server nodes. For each group of 64 CNs (called a *pset*), there is a single ION that receives I/O requests from the CNs in that group and forwards those requests over its 10 gigabit ethernet port to the external storage system. In total, 640 IONs are connected through 10 Gigabit Ethernet ports to the storage system, which consists of a diameter-5 Myricom Myri-10G switch complex. At the other end of the switch complex are 128 storage servers. These servers are attached via InfiniBand 4X DDR to 16 Data Direct Network 9900 storage devices (or just *DDN 9900*). The DDN 9900 performs RAID 6 parity calculations, data integrity validation, and online drive rebuilds, as well as providing redundant paths to its storage. The DDN 9900 exports block devices as logical units (*LUN*s). Each DDN 9900 contains two controllers with four InfiniBand ports each that receive SCSI requests to the LUNs over InfiniBand. The Intrepid system is configured so that each LUN is exported across all the ports on each controller, providing eight servers with redundant device access. This allows up to seven of the nodes connected to a single DDN 9900 to fail, while still providing availability to the LUNs exported by the DDN 9900. Also, if one of the controllers on the DDN 9900 fails, data for the LUNs is still available through the other controller. This combination of one DDN 9900 with two controllers, eight ports, and eight storage nodes makes up a redundancy group that tolerates failures and allows for the software environment to provide high-availability guarantees to the storage. The entire storage system consists of 16 of these redundancy groups, with 128 storage nodes in total.

3.2 I/O Software

Multiple software layers are involved in the I/O path. Applications use I/O libraries, such as HDF5, or PnetCDF, or may call MPI-IO or POSIX system calls directly. If MPI-IO is used (either by the application or a higher-level library), MPI-IO optimizations such as two-phase I/O are accomplished via communication over the 3D torus network between compute nodes.

The BG/P system allows applications to run in *virtual node mode*, where a separate process runs on each of the four cores of a CN. In the Intrepid configuration, with 64 CNs per ION,

each ION may perform up to 256 independent I/O requests to the file system at once. From the file system perspective, this results in as many as 163,840 independent processes performing I/O requests concurrently from 640 distinct client nodes. All requests to the file system from CN processes are converted into one or more POSIX-like operations, forwarded over the internal tree network, and received by an I/O forwarding daemon, called the *ciod*, running on an ION. IONs run the Linux operating system and mount external file systems. The ciod forks a separate process to represent each CN process in its pset (up to 256 total). These processes replay system calls made by their corresponding CN process. These calls make their way through the Linux kernel virtual file system layer and are subsequently processed by the specific file system, in our case PVFS. PVFS is an open source parallel file system designed for computational science applications [17, 18, 19, 20]. The PVFS client software running on the ION receives system calls from the kernel and translates these operations into requests to be sent over the Myricom network to the PVFS metadata and I/O servers running on the storage nodes. PVFS servers in turn receive requests and perform operations to the InfiniBand connected storage devices. In addition to a Linux kernel module that provides a POSIX-like interface, PVFS provides a rich interface for describing I/O accesses. Support for this interface is provided in the ROMIO MPI-IO implementation [21]; when used, this interface significantly improves performance for certain classes of HPC applications [22, 23]. Because of the BG/P dependence on the IBM I/O forwarding implementation, all PVFS accesses must be vectored through the Linux kernel on IONs, preventing the use of the richer PVFS interface in this deployment. However, a number of other features in the I/O software enable scalable and resilient operation in this demanding environment.

3.2.1 Collective I/O

On Intrepid, the MPI-IO layer can perform *collective I/O*, giving a subset of the processes the responsibility of performing I/O on behalf of all the processes. This allows for aggregation, transforming many small I/O accesses (a common workload for many applications) into several larger I/O accesses (a more optimal workload for I/O systems). I/O operations that would otherwise appear out of order to the storage device can be aggregated and performed as a single operation or as fewer in-order operations. Aggregation also allows I/O to be aligned to the blocksize of the filesystem, which has shown significant performance improvement in parallel file systems [2]. On Blue Gene systems such as Intrepid, the dedicated I/O forwarding nodes effectively limit the number of processes that achieve peak performance from the I/O system [24]. Hence, collective I/O is a good choice for these systems.

Leadership-computing systems today often have low-latency communication networks that are separate from the high-bandwidth (but often higher-latency) I/O network. Collective I/O is able to take advantage of this imbalance in latency between the compute nodes and the storage devices, performing smaller, latency bound I/O operations over the communication network to the designated I/O nodes, which only perform larger, bandwidth intensive I/O operations to storage. Further, if separate communication and I/O networks do exist on the system, collective I/O is able to augment the performance of the storage hardware by exploiting the fast communication networks on these machines.

3.2.2 Consistency

The PVFS file system deployed on Intrepid relaxes the POSIX consistency semantics [25], which require synchronization between processes performing I/O operations to the same file. Computational science applications often exhibit coordinated, fine-grained sharing of datasets [26], eliminating the need for POSIX consistency.

File systems that enforce POSIX consistency semantics often use mechanisms that hinder the performance of writes to distinct regions of a file, unless those writes are carefully aligned with internal file system boundaries [2]. As a result, HPC systems exhibit poor performance at higher process counts [12] and confusing I/O system behavior, forcing computational scientists to split I/O from individual processes into separate files [16]. Instead of enforcing POSIX consistency, PVFS defines simultaneous accesses to *overlapping byte regions* of a file to have undefined results.[1]

3.2.3 State avoidance

Many parallel file systems cache data at the client on behalf of the application. Keeping this state on the client allows the file system to perform larger accesses to the storage servers, as well as improving locality. Caching is appropriate for systems with applications that have serial workloads writing to separate files, but it has reduced benefit and often hinders performance on systems with huge client nodes counts or on systems with little memory available for caching. On the IBM Blue Gene/P, only the ION can perform caching on behalf of the CNs in its pset. With a CN to ION ratio of 64 to 1, only a small portion of memory is available for caching. Cache thrashing is likely and can hinder I/O performance.

On systems without I/O forwarding, where I/O is performed directly by the client nodes, caching at the client requires locking I/O accesses. For systems with tens of thousands of active clients, the overhead of locking becomes a performance bottleneck, particularly in concurrent I/O scenarios, such as when a large application run attempts to checkpoint [2].

Statelessness is important for the file system protocol as well. Clients using a stateless protocol are not required to support callback notifications to revoke held locks or do not require that servers monitor client status. This feature is important on systems where lightweight client kernels running on system nodes do not support multiple threads or where the cost of threads devoted to receiving notifications is prohibitive. As systems have gotten larger, the increasing number of independent components has dramatically increased the likelihood of failure of a single component; and with stateful clients, failures of a single client for a large run can bring all I/O to a halt. Keeping state within the

[1] The I/O forwarding software provided by IBM also relaxes the POSIX write semantics, by splitting up I/O operations. Enforcing them at the file system layer on a Blue Gene/P system results in lower performance for concurrent accesses without significantly stronger semantic guarantees than those provided by PVFS.

client requires separate lock manager nodes to manage the shared state at the clients. These lock managers introduce additional management overhead and points of failure to the overall system. By contrast, PVFS implements a stateless design, where clients (BG/P IONs) push data to servers immediately, instead of attempting to cache data on behalf of clients. Open calls translate to a single lookup at one of the metadata servers, and close calls are a local operation that don't perform any server requests. This design allows I/O operations to perform I/O requests directly to the I/O servers, bypassing metadata servers or lock managers and eliminating unnecessary round trips that increase latency.

3.2.4 Distributed metadata

PVFS incorporates a collection of optimizations [17] designed to enable high performance for non traditional application access patterns, such as metadata-heavy workloads. This is motivated by emerging HPC application domains that perform in parallel many small-file operations and have increasingly intensive metadata workloads [27]. PVFS features that support these workloads include the ability to distribute metadata across all servers, concise metadata representations for small files, and optimizations to lower latency for small file accesses.

4. PERFORMANCE EVALUATION

Our evalation of the scalability of the I/O architecture on Intrepid includes systematic testing of each level of the storage system. First we perform tests of individual components in the storage system. Then we perform synthetic benchmarks of the entire I/O subsystem, scaling from 2,048 to 131,072 processors. For large-scale application runs, we choose from a collection of benchmarks that measure the behavior of the storage system under various workloads, performing large-scale runs of those benchmarks and analyzing the performance results.

4.1 Experimental Setup

The experiments were performed during an acceptance period before the machine and file system were in a production mode. Our experiments used PVFS 2.8.0 (it did not include some minor fixes that were added prior the release). PVFS distributes both file data and metadata across multiple storage servers, based on the chosen configuration. For our experiments we configured a PVFS volume that consisted of 123 servers capable of handling both metadata and I/O requests. Groups of 8 servers were accessing storage devices on each of the 16 DDN 9900s (to allow for a few failures during testing, we withheld one DDN 9900 controller and one storage node from the pool of storage). The PVFS servers were configured to access storage devices using direct I/O to bypass the Linux kernel buffer cache.[2] This configuration was chosen because it demonstrated significant performance improvement for large accesses to the DDN 9900. Unless otherwise noted, the bandwidth measurements throughout our experiments are given in the IOC standard units of mebibytes (2^{20}) or gibibytes (2^{30}), abbreviated *MiB* and *GiB*, respectively. The servers were configured with a stripe unit of 4 MiB, chosen to match the maximum buffer size used by the ciod for each file system operation.

[2]This was enabled in the PVFS config file with `TroveMethod directio`.

The results were gathered after aggressively tuning each component of the I/O stack and were taken during an acceptance period where the system was unavailable to users and provided consistent performance. While our goal was to measure the overall performance of the system for applications running during production, large variations in performance from week to week and even day to day were seen, most often due to storage component failures and firmware upgrades. Our experience suggests that reproducing results is extremely challenging on systems of this complexity.

4.2 Component Analysis

To establish the I/O rates the machine is capable of sustaining through the IBM ciod I/O forwarding framework, we performed a set of tests moving data between CNs and IONs only. Figure 2(a) shows the available bandwidth between CNs and IONs on BG/P using either 1 or 64 compute processes and a 4 MiB block size. We chose the 4 MiB block size to reflect the maximum request size that can be produced by the I/O forwarding software. The bandwidth is measured by performing reads and writes to the */dev/zero* special file in order to eliminate parallel file system costs. Reads require multiple processes to saturate the link because of the overhead of providing zeroed out buffers on the ION side. For writes, a single process is enough to saturate the link.

Figure 2(b) shows the bandwidth between IONs and storage nodes as measured with NetPipe. The IONs are able to drive the external 10 Gigabit Ethernet network at only approximately 276 MiB/s. This ION limit defines the maximum rate at which a set of CNs associated with an ION will be able to access storage. Also, the ION performance showed irregularity at a number of intervals between message sizes of 16,384 to 262,144 bytes. In order to show complete performance, NetPipe perturbs the message sizes sent so that not all messages are multiples of 2. We see that for a few of these unaligned small messages, performance is severely reduced. This may have a dramatic effect on applications that perform smaller I/O with unaligned message sizes.

Figure 3 shows the peak I/O rates attained when accessing the DDN attached storage using one to seven storage servers with different numbers of threads. To mimic large I/O patterns, we performed large contiguous direct accesses of 4 MiB to the device using *sgp_dd*. We did not see an improvement in performance with all 8 servers accessing the device, and so we deployed the production file system with only seven servers per storage device. These results were taken after the storage devices had been configured for the production configuration.

For the Intrepid configuration, PVFS stores metadata on the DDN 9900 storage device. Because of this setup, we wanted to measure the cost of metadata updates, which occur locally for many write operations (writes past end-of-file require an update to the size). The metadata updates consist of small (4,096 to 262,144) writes, followed by a sync operation. The performance of the device for these small write+sync operations is in the range of 5-8 MiB/s, far less than for the larger I/O accesses. This results in an added cost to write operations, reducing peak write bandwidth for a single storage node from 600 MiB/s to roughly 550 MiB/s. Performance

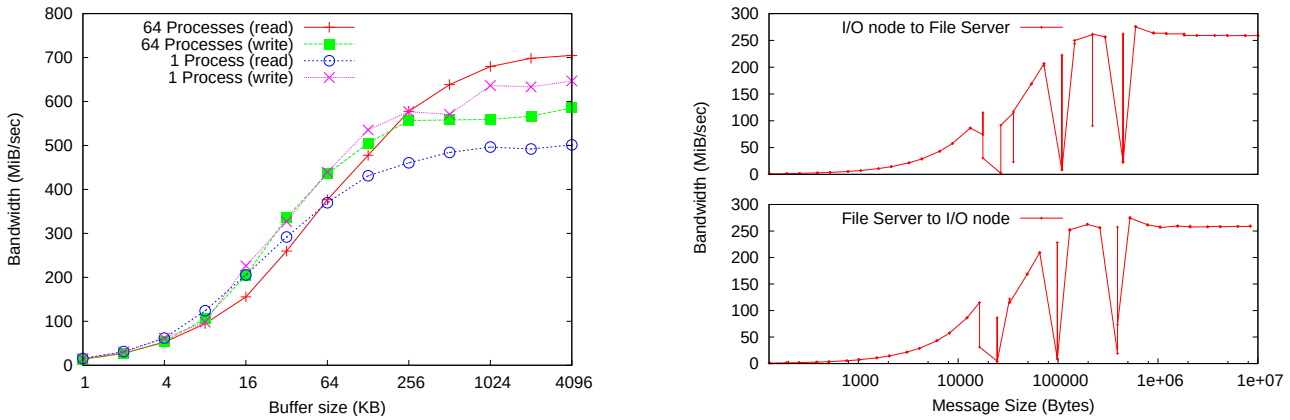

Figure 2: Throughput between CN and ION, accessing *dev/zero* (left). Bandwidth between ION and file servers (right).

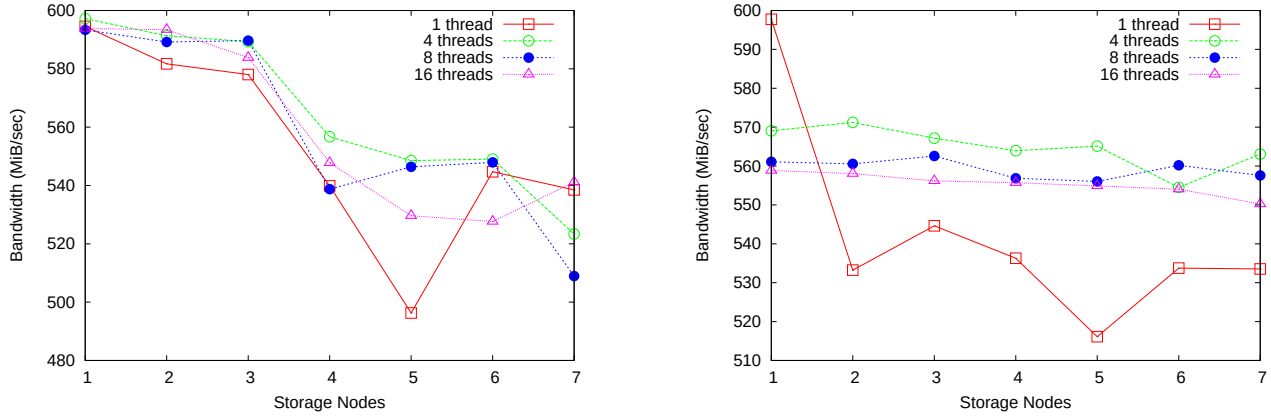

Figure 3: Per-node throughput of the DDN 9900 storage device reading (left) and writing (right).

of applications that have metadata-intensive workloads are limited by this rate as well.

To determine the aggregate peak rate of the external ethernet, we configured the PVFS servers to *not* perform I/O to the storage devices,[3] and we measured the performance of large collective reads and writes at 131,072 processes. For these tests, the aggregate write bandwidth of the network maintained a peak rate of 45.19 GiB/s and a peak read bandwidth of 59.87 GiB/s. We were unable to pinpoint the cause of the discrepancy between reads and writes, but we speculate that with 512 nodes on the client side of the network and 123 nodes on the server side a variation of the incast problem [28] limits the aggregate write bandwidth.

Combined, these results determine the overall peak I/O rate of the system for varying numbers of processes. They show that the I/O system has a reasonably balanced provisioning, where storage component peak rates are reached only when applications perform runs utilizing almost half of the machine.

[3]This was enabled in the PVFS configuration file with `TroveMethod null-aio`.

Using these component peak rates, we can calculate the aggregate peak rate of the system as shown in Figure 4. From the storage side, we should not expect to exceed 65.9 GiB/s aggregate read or 68.3 GiB/s aggregate write performance. The external network further limits aggregate write bandwidth to 45.19 GiB/s. From the ION side, the per-ION rate limits the maximum bandwidth for reading until 244 IONs are in use (62,464 client processes in virtual node mode). For writing, the maximum bandwidth is limited by the per-ION rate until 285 IONs are in use (45,116 client processes).

4.3 Scaling Analysis

4.3.1 IOR

We employ the IOR benchmark from LLNL to measure aggregate I/O throughput. IOR is a flexible, synthetic I/O benchmark that allows I/O through a variety of interfaces and coordinates timing and access using MPI communication. The coordination of timing is critical when measuring performance with very large numbers of clients.

Figures 5(a) and 5(b) show the IOR throughput for reads and writes, respectively, using the POSIX interface. The aligned tests used 4 MiB accesses for a total of 64 MiB per

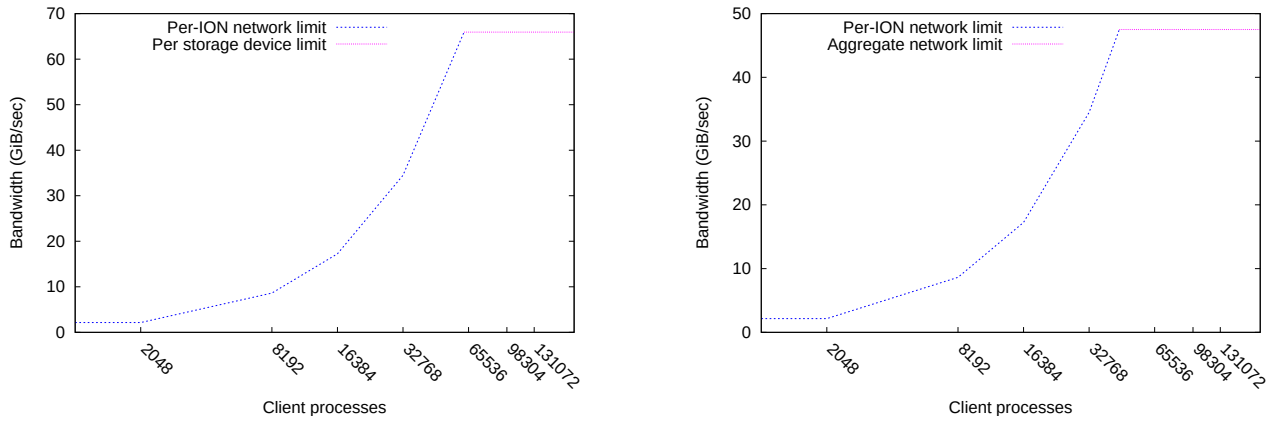

Figure 4: Peak aggregate throughput for read (left) and write (right) of the newtork and storage devices at varying process counts.

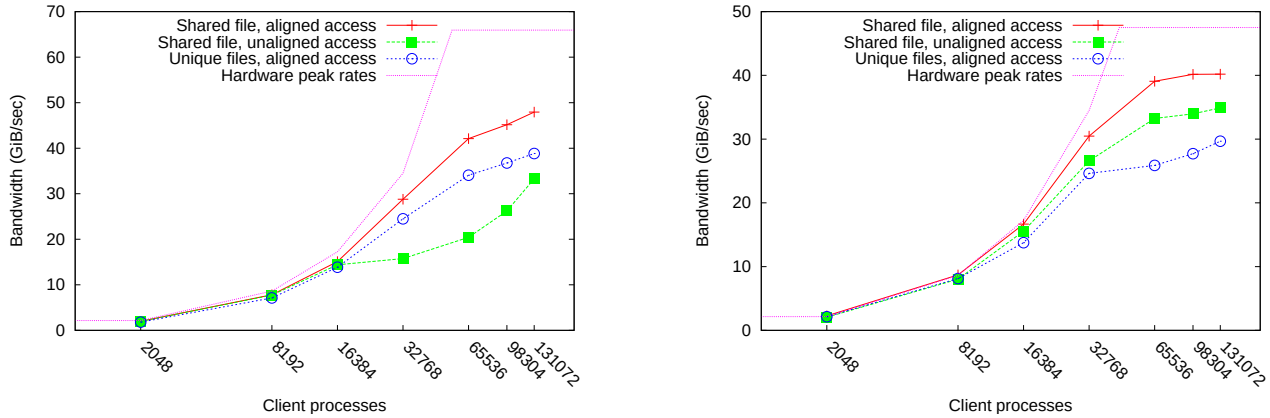

Figure 5: IOR aggregate read (left) and write (right) performance with 64 MiB per process.

process. With aligned accesses, each process performs accesses that align perfectly with the stripe of the file, so the large 4 MiB accesses are made directly to the storage device. The unaligned tests used 4 MB (4×10^6 bytes) accesses for a total of 64 MB per process. With unaligned accesses, the requests span multiple file stripe units, requiring that requests be serviced by two storage nodes rather than one.

The results show that performance was best with shared file, aligned accesses. For writes, performance begins to level off at 64K processes and remain constant. At a maximum rate of 40.2 GiB/s for writing, we were able to achieve roughly 85% of the peak rate for this hardware configuration. Read performance continues to increase with the process count, reaching a maximum rate of 47.9 GiB/s at 131,072 processes, roughly 79% of the peak hardware rate. These results show that performance is maintained at the highest processor counts and that the storage system is able to scale with the size of the application. The results also demonstrate inefficiency in the storage hardware and software and provide insight into some of the challenges of achieving peak performance in leadership storage systems.

At large process counts there is a deviation from the peak hardware rate. As larger application sizes place heavier workloads on the storage nodes, we notice deficiences in both the storage device and file system software at the storage node. Two notable file system deficiencies stand out. First, at higher process counts we see roughly a 20-25% decrease in performance with unique files compared to shared files. This can be attributed to the PVFS data placement algorithm, which uniformly distributes a file over all the storage nodes but chooses randomly the first node to begin striping. When shared file accesses are performed to a single file, I/O is balanced evenly over the storage nodes. However, with file-per-process workloads, we see hot spots develop at the storage nodes, decreasing the overall performance of the application.

Second, the file system does not aggregate I/O requests to the storage device. For unaligned and unique file workloads, the storage device sees many concurrent smaller requests, often out of order. Few storage devices perform well under this workload; and although the DDN 9900 device aggregates requests together, a decrease in performance is seen with smaller concurrent I/O requests. This behavior is most no-

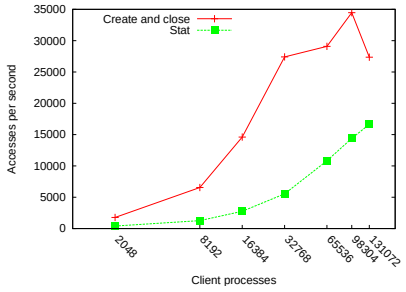

Figure 6: Metarates create/close and stat rates, varying number of client processes.

ticeable in the comparison of aligned and unaligned accesses. With aligned accesses, a single 4 MiB request is performed from an ION to a storage node; but with unaligned accesses, an I/O request is split into two uneven requests to different storage nodes. The resulting throughput is limited by the smaller I/O request to the storage device. The DDN 9900 device provides a *write-caching* mode, which was enabled for these tests. This allows the device to perform more aggregation for heavy write workloads than for reading, which hides some of the unaligned performance drop in the case of writing. Moreover, when writing unique files, each I/O request at the storage node requires a metadata update to the size of the data object. These updates require 4 KiB write requests to the storage device. This requirement limits the throughput when writing unique files to the storage device performance for 4 KiB accesses. PVFS coalesces these small updates, but the coalesced accesses remain relatively small, and the difference is marginal at higher process counts.

4.3.2 Metarates

While file system create and file stat rates are not often a critical factor in computational science applications, they can be important in other domains [27]. Here we demonstrate scalability of the file system that goes beyond the requirements of the HPCS PERCS program, including creating 32,000 files per second and performing 10,000 metadata operations per second.

The metarates benchmark from University Corporation for Atmospheric Research (UCAR) performs metadata operations using POSIX system calls (`creat` and `stat`), using MPI barrier operations to synchronize between processes. Figure 6 shows the total rate to both create and close new files as well as the rate to stat existing files. The number of clients is scaled as in the IOR case. The number of servers is fixed at 123. Each process accesses 2 files, for a total of up to 262,144 files accessed at the largest data point. Each client operated in its own subdirectory to prevent synchronization at the metadata server managing a single directory.

These results demonstrate create and stat rates scaling with the number of processes, performing 34,470 create operations per second at 98,304 processes. The stat rate does not scale up at the same rate but still shows a stat rate of 16,642 operations per second at 131,072 processes. We attribute the differences in create and stat rates to the performance of the storage device for small I/O accesses. Creating a file requires writing a 4 KiB page to the storage device, while stat requires reading a 4 KiB page from the storage device. In this case, we see that the storage device performs better with small write requests than with reads.

Table 1: BTIO problem size E, 65,536 processes

Performance	Measurement
I/O timing	91.73 seconds
Percent I/O	30.35%
Total data written	1976.65 GiB
I/O data rate	21.55 GiB/s

4.4 Application Benchmarks

The results presented so far give us an indication of the peak performance possible using the combination of hardware and file system software deployed. However, the success of this I/O system is defined by its ability to provide performance for applications, not for synthetic benchmarks. In this section we investigate how the system performs using three benchmarks that simulate I/O from applications that exhibit weak scaling in three different scientific domains. To capture the performance of a typical application running on the system, we chose not to tune the benchmarks to improve their performance. Instead, we inspect the I/O patterns of each benchmark run with their default settings and discuss some of the causes for seeing reduced performance from our IOR scaling study measurements. To understand the I/O patterns of the application benchmarks, we used Darshan [29], a tool developed at Argonne to inspect application I/O workloads as the application runs.

4.4.1 BTIO

BTIO is an I/O-only version of the BT (block tridiagonal) benchmark found in the NAS Parallel Benchmarks (NPB) collection [30]. The BT benchmark performs a complex domain decomposition across a square number of nodes, where each compute node is responsible for multiple Cartesian subsets of an entire dataset. The BTIO benchmark performs large collective MPI-IO writes and reads of a nested strided datatype and is an important test of the performance a system can provide for noncontiguous workloads. The BTIO benchmark was built from NPB version 3.3 with the "full" subtype, which includes a Class E problem size. Class E corresponds to a grid size of $1020 \times 1020 \times 1020$, which writes in aggregate 1.93 TiB of data to a shared file every fifth timestep using 65,536 processes. Table 1 shows the results of running BTIO on Intrepid.

BTIO reports overall throughput, so the aggregate throughput of 21.55 GiB/s is a measure of both the write and read phases of the benchmark. This matches closely with what we see for read performance at 65,536 processes in our IOR measurements. The BTIO benchmark uses MPI-IO collective writes and reads to a shared file and, in "full" mode, enables the MPI-IO collective buffering optimization, which designates a subset of compute nodes that perform I/O to the file system. For this run, the BTIO default collective buffer size of 1 MB (1×10^6) was used. The number of collective buffering nodes on Intrepid is set to 8 per ION, resulting in a total of 2,048 CNs performing I/O through 256 IONs at

1 MB contiguous requests. With a 1 MB collective buffer, 95% of the I/O performed by BTIO was unaligned. This result shows the importance of tuning the application and I/O interfaces to the unique characteristics of the storage system. Based on results from our IOR scaling measurements, we expect a substantial improvement if buffers are aligned by the BTIO application.

4.4.2 MADbench2

MADbench2 is a benchmark derived from the MADspec data analysis code. The MADspec code estimates the angular power spectrum of cosmic microwave background radiation in the sky from noisy pixelized datasets. As part of its calculations, the MADspec code (and likewise the MADbench2 benchmark) performs extremely large out-of-core matrix operations, requiring successive writes and reads of large contiguous data from either shared or individual files. Because of the large, contiguous mixed read and write patterns that MADbench2 performs and its capabilities to test a variety of parameters (shared files, POSIX vs. MPI-IO, etc.), it has become a leading benchmark in the parallel I/O community [14, 31].

In our tests, we examine the scalability of the file system with many processes reading and writing to a shared file, an I/O pattern that becomes increasingly difficult at scale for many file systems but is necessary in data analysis applications where the number of processors may change from one run to the next [14]. We ran MADbench2 with 65,536 processes (MADbench2 requires a perfect square number of processes). MADbench2 was compiled to run in I/O mode, so our tests did not include MPI communication or perform significant computation (the busy-work exponent α was set to 1). Thus, these results demonstrate just the capability of the file system and underlying storage architecture. The file alignment used by MADbench2 for these runs was the default of 4,096. We allowed all processes to perform I/O simultaneously (*RMOD* and *WMOD* both set to 1) and fixed the problem size at *NPIX = 400,384* so that each process was performing I/O operations of roughly 20 MiB. In aggregate, the I/O performed by each component of the benchmark totaled 5.13 TiB.

The results of the MADbench2 run are shown in Figure 7. Because of the large contiguous I/O performed by MADbench2, we were able to achieve roughly 45 GiB/s writing and 27 GiB/s reading. This performance difference between writing and reading is directly related to the mixed nature of the MADbench2 I/O workload. Before and after each I/O operation, MADbench2 performs a barrier, synchronizing all processes to wait for completion of the slowest I/O operation, and forcing read and write operations into nonoverlapping steps. At the storage device, a large writeback cache allows the write operations to become effectively network limited, resulting in very fast responses to all write operations. Because the MADBench2 code performs a mixed workload, the read step that follows a write step provides a window of time for the storage devices to flush their writeback cache to disk. This results in higher write performance overall than what is seen in the IOR scaling measurements, which perform many consecutive writes, preventing the writeback cache from improving performance.

Read performance does not get the same benefit of the write-back cache at the storage device, and we see performance is in line with the IOR scaling measurements. Because the MADbench2 run used the default filesystem block size of 4,096, many of the read operations were unaligned. As a result, we see that MADbench2 performance for reads falls between the purely aligned and entirely unaligned IOR performance.

4.4.3 Flash3 I/O

The FLASH code was developed to simulate compressible reactive flows that occur in astrophysical environments. It has been shown to be highly scalable; and has an I/O-only mode for measuring the I/O performance of FLASH on a deployed system. A case study of FLASH3 on Lawrence Livermore's BG/L machine performed runs at up to 64K processes [16]. Because of the performance degredation of shared-file I/O, the authors were forced to write a file per process and were then met with the challenge of managing and postprocessing the nearly 75 million files generated. Their study demonstrates the scalability problems that often arise in parallel file systems from shared file access.

The FLASH3 package provides an I/O kernel with unit tests that mimic the I/O workloads of the full FLASH3 simulation. In our experiments, we performed FLASH3 I/O kernel runs using both the PnetCDF and HDF I/O interfaces (both write to shared files), at a scale of 65,536 processors using 16,384 nodes in virtual node mode. Each run consisted of writing a single checkpoint file and plot file. We used a uniform grid with $150 \times 150 \times 150$ grid points with 6 variables. This resulted in a checkpoint file of roughly 14TiB and a 2TiB plot file.

The results of our FLASH3 runs are shown in Figure 7. The FLASH3 I/O workload (for both the checkpoint and plot files) consists of writing to a shared file with large unaligned accesses. Performance of the checkpoint files for both HDF5 and PnetCDF was just above 20 GiB/s. This is less than our IOR scaling measurement for the same number of processes in the aligned, shared file case. We investigated the I/O patterns of the FLASH3 code for both HDF and PnetCDF while writing the checkpoint files and found that with PnetCDF all I/O used MPI-IO collective I/O routines, while with HDF5 only independent I/O was used. Because HDF5 performs independent I/O at each process, most of the writes are unaligned, and we see many small writes (less than 100 bytes) from HDF5 writing the checkpoint file, resulting in considerably reduced performance. With PnetCDF, the collective calls require processes to first send regions to I/O aggregators, which then perform the write in large, contiguous chunks, synchronizing all processes until the write completes. Thus, in the PnetCDF checkpoint result, we see the overhead of buffering at the aggregators and synchronizing all processes for each write, which is roughly the same as the overhead of performing many small, unaligned writes independently at each process (with HDF5). We noticed a significant performance drop for writing the HDF5 plotfile and, upon investigating the I/O workload, found that the HDF5 interface in the FLASH3 I/O code unit does not enable collective I/O.[4] Also, HDF5 performs many very small writes

[4] HDF5 collective I/O is included in FLASH3 I/O code only when

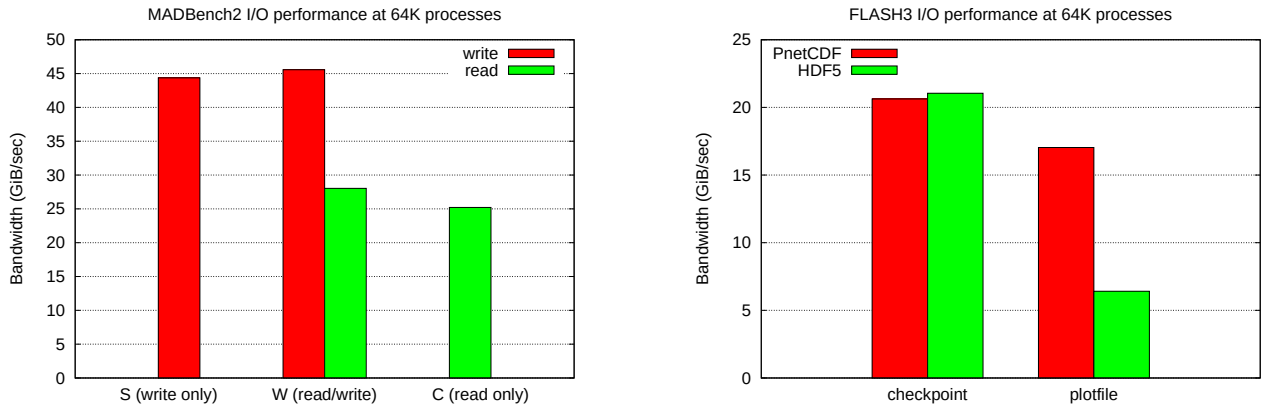

Figure 7: Application I/O benchmarks: MADbench2 (left) and FLASH3 (right).

(less than 100 bytes each), most likely to update metadata. Taken together, these two findings explain the poor HDF5 plotfile performance.

These results show that performance of an application's I/O workload depends highly on the I/O interfaces in use and on the modes specified to those interfaces. We would expect HDF5 plotfile performance to improve if the collective I/O mode were enabled for that interface, but further investigation of the workload differences between these two I/O interfaces is necessary.

5. CONCLUSIONS AND FUTURE WORK

In this study, we described a complete I/O system including the hardware and software, deployed at a leadership-facility, and we discussed how the components work together to provide I/O services to applications. We performed a comprehensive study of I/O performance, starting with individual components and building up to system-wide application I/O simulations. Our component analysis allowed us to assess the balance of the hardware resources in the system, while the comprehensive and application-oriented analysis enabled us to understand how effectively the components are utilized in real-world applications.

We found that the I/O system on Intrepid holds up well to the I/O demands of large-scale applications but that reaching peak performance of the I/O hardware is increasingly challenging at the highest process counts. With increasing relative cost to the I/O subsystem for petascale machines, I/O is quickly becoming the bottleneck for many computational science applications. It is essential,therefore, that the I/O subsystem perform near-peak rate for the largest of runs.

We also found that application I/O performance varies considerably, not only because of application's I/O patterns, but also because of the different I/O libraries and tuning parameters used for a particular run.

Some of the challenges facing leadership-computing systems today are the same challenges of the past. Our results demonstrate that systems continue to struggle with the difference between the large, contiguous, aligned I/O patterns that get the best performance from the I/O system and the smaller, noncontiguous, unaligned I/O patterns of computational science. Meeting these challenges will require a combined effort from storage system designers, file system developers, and application writers. Storage systems of tomorrow need to handle wide variations in the I/O patterns and utilization of the storage system. File systems and I/O libraries must be developed to perform well under many different I/O workloads. Application writers must become aware of these I/O challenges and take advantage of the optimizations and tuning parameters provided by I/O software.

Leadership systems face new challenges as well. Unprecedented process counts and storage components increase the complexity of the storage system and the likelihood of component failures. Moreover, applications running near-full capacity on these systems quickly uncover scalability problems in file system design. Storage systems will need to handle component faults seemlessly, but file systems must also be designed to avoid single points of contention, for both metadata and I/O accesses.

While previous HPC system studies performed scalability measurements with less than 50% of the machine, we have seen that on a leadership-system with balanced provisioning, performance measurements are often necessary at higher process counts to discover the interplay between the network and storage capacity. On Intrepid, I/O bandwidth is effectively partitioned by IONs; only the largest jobs have the potential to monopolize the I/O system, whereas users running at smaller process counts can expect I/O performance to scale roughly with the size of the job.

We will continue to study the behavior of the I/O system on Intrepid. Future work will focus on characterizing system performance over an extended period of time. As the production system is used by computational science, changes to underlying software and hardware will cause performance changes, which we intend to characterize. Emphasis will be placed on performance and resilience in the case of failure,

compiled with the COLLECTIVE_HDF5 macro defined.

to understand both the effects on performance during failure recovery, and the way performance degrades in different failed states.

6. ACKNOWLEDGMENTS

We are thankful to Kamil Iskra of Argonne National Laboratory for his help in understanding the performance characteristics of the Blue Gene/P I/O nodes and tree network.

This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

7. REFERENCES

[1] W. Yu, J. Vetter, R. Canon, and S. Jiang, "Exploiting Lustre file joining for effective collective I/O," in *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid, 2007. CCGRID 2007*, 2007, pp. 267–274.

[2] W. Liao and A. Choudhary, "Dynamically adapting file domain partitioning methods for collective I/O based on underlying parallel file system locking protocols," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, Piscataway, NJ, 2008.

[3] F. Schmuck and R. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *Proceedings of the FAST 2002 Conference on File and Storage Technologies*, January 2002.

[4] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable performance of the Panasas parallel file system," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, February 2008.

[5] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: a scalable, high-performance distributed file system," in *OSDI '06: Proceedings of the 7th symposium on Operating Systems Design and Implementation*. Berkeley, CA: USENIX Association, 2006, pp. 307–320.

[6] R. Hedges, B. Loewe, T. McLarty, and C. Morrone, "Parallel file system testing for the lunatic fringe: the care and feeding of restless I/O power users," in *Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies*, April 2005.

[7] W. Yu, S. Oral, J. Vetter, and R. Barrett, "Efficiency evaluation of Cray XT parallel I/O stack," in *Cray Users Group Meeting*, 2007.

[8] R. Oldfield, L. Ward, R. Riesen, A. Maccabe, P. Widener, and T. Kordenbrock, "Lightweight I/O for scientific applications," in *Proceedings of the IEEE International Conference on Cluster Computing*, 2006, pp. 1–11.

[9] M. Oberg, H. Tufo, and M. Woitaszek, "Exploration of Parallel Storage Architectures for a Blue Gene/L on the TeraGrid," in *9th LCI International Conference on High-Performance Clustered Computing*, March 2008.

[10] J. Laros, L. Ward, R. Klundt, S. Kelly, J. Tomkins, and B. Kellogg, "Red Storm IO Performance Analysis," *Cluster, Austin, TX*, 2007.

[11] W. Yu, J. S. Vetter, and H. S. Oral, "Performance characterization and optimization of parallel I/O on the Cray XT," in *Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium*, April 2008.

[12] M. Fahey, J. Larkin, and J. Adams, "I/O performance on a massively parallel Cray XT3/XT4," in *International Symposium on Parallel and Distributed Processing, 2008*, April 2008, pp. 1–12.

[13] M. Kandaswamy, M. Kandemir, A. Choudhary, and D. Bernholdt, "Performance implications of architectural and software techniques on I/O-intensive applications," in *International Conference on Parallel Processing, 1998*, 1998, pp. 493–500.

[14] J. Borrill, L. Oliker, J. Shalf, and H. Shan, "Investigation of leading HPC I/O performance using a scientific-application derived benchmark," in *Proceedings of Supercomputing*, November 2007.

[15] H. Shan, K. Antypas, and J. Shalf, "Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark," in *Proceedings of Supercomputing*, November 2008.

[16] R. T. Fisher, L. P. Kadanoff, D. Q. Lamb, A. Dubey, T. Plewa, A. Calder, F. Cattaneo, P. Constantin, I. Foster, M. E. Papka, S. I. Abarzhi, S. M. Asida, P. M. Rich, C. C. Glendening, K. Antypas, D. J. Sheeler, L. B. Reid, B. Gallagher, and S. G. Needham, "Terascale turbulence computation using the FLASH3 application framework on the IBM Blue Gene/L system," *IBM J. Res. Dev.*, vol. 52, no. 1/2, pp. 127–136, 2008.

[17] P. Carns, S. Lang, R. Ross, M. Vilayannur, J. Kunkel, and T. Ludwig, "Small-file access in parallel file systems," in *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium*, April 2009.

[18] P. Carns, W. Ligon III, R. Ross, and R. Thakur, "PVFS: A parallel file system for Linux clusters," in *Proceedings of the 4th Annual Linux Showcase & Conference, Atlanta-Volume 4*. USENIX Association Berkeley, CA, 2000, pp. 28–28.

[19] A. Ching, A. Choudhary, W. Liao, R. Ross, and W. Gropp, "Noncontiguous I/O through PVFS," in *Proceedings of the 2002 IEEE International Conference on Cluster Computing*, 2002, pp. 405–414.

[20] D. Hildebrand and P. Honeyman, "Exporting storage systems in a scalable manner with pNFS," in *Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies*, 2005, pp. 18–27.

[21] R. Thakur, W. Gropp, and E. Lusk, "Data sieving and collective I/O in ROMIO," Mathematics and Computer Science Division, Argonne National Laboratory, Tech. Rep. ANL/MCS-P723-0898, August 1998. [Online]. Available: http://www.mcs.anl.gov/ thakur/papers/romio-coll.ps

[22] R. Latham, R. Ross, and R. Thakur, "The impact of file systems on MPI-IO scalability," *Lecture Notes in Computer Science*, pp. 87–96, 2004.

[23] A. Ching, A. Choudhary, W. K. Liao, R. Ross, and W. Gropp, "Evaluating structured I/O methods for

parallel file systems," *International Journal of High Performance Computing and Networking*, vol. 2, pp. 133–145, 2004.

[24] H. Yu, R. Sahoo, C. Howson, G. Almasi, J. Castanos, M. Gupta, J. Moreira, J. Parker, T. Engelsiepen, R. Ross *et al.*, "High performance file I/O for the Blue Gene/L supercomputer," in *Proceedings of the Twelfth International Symposium on High-Performance Computer Architecture*, 2006, pp. 187–196.

[25] IEEE, *1003.1-1988 INT/1992 Edition, IEEE Standard Interpretations of IEEE Standard Portable Operating System Interface for Computer Environments (IEEE Std 1003.1-1988)*. New York, NY: IEEE, 1988.

[26] D. Kotz and N. Nieuwejaar, "Dynamic file-access characteristics of a production parallel scientific workload," in *Proceedings of Supercomputing'94*, 1994, pp. 640–649.

[27] D. Koester, "What makes HPC applications challenging?" in *Workshop on Patterns in High Performance Computing*, May 2005.

[28] A. Phanishayee, E. Krevat, V. Vasudevan, D. Andersen, G. Ganger, G. Gibson, and S. Seshan, "Measurement and analysis of TCP throughput collapse in cluster-based storage systems," Carnegie Mellon University, Tech. Rep. CMU-PDL-07-105, September 2007.

[29] Philip Carns and Robert Latham and Robert Ross and Kamil Iskra and Samuel Lang and Katherine Riley, "24/7 Characterization of Petascale I/O Workloads," in *Proceedings of the First Workshop on Interfaces and Abstractions for Scientific Data Storage*, New Orleans, LA, September 2009.

[30] P. Wong and R. der Wijngaart, "NAS parallel benchmarks I/O version 2.4," no. NAS-03-002, 2003.

[31] J. Borrill, J. Carter, L. Oliker, D. Skinner, and R. Biswas, "Integrated performance monitoring of a cosmology application on leading HEC platforms," in *Proceedings of the International Conference on Parallel Processing*, June 2005.