On the Role of Burst Buffers in Leadership-Class Storage Systems

Ning Liu,*† Jason Cope,† Philip Carns,† Christopher Carothers,*
Robert Ross,† Gary Grider,‡ Adam Crume,§ Carlos Maltzahn,§

*Rensselaer Polytechnic Institute, Troy, NY 12180, USA
{liun2,chrisc}@cs.rpi.edu
†Argonne National Laboratory, Argonne, IL 60439, USA
{copej,carns,rross}@mcs.anl.gov
‡Los Alamos National Laboratory, Los Alamos, NM 87545, USA
ggrider@lanl.gov
§University of California at Santa Cruz, Santa Cruz, CA 95064, USA
{adamcrume,carlosm}@soe.ucsc.edu

***Abstract*—The largest-scale high-performance (HPC) systems are stretching parallel file systems to their limits in terms of aggregate bandwidth and numbers of clients. To further sustain the scalability of these file systems, researchers and HPC storage architects are exploring various storage system designs. One proposed storage system design integrates a tier of solid-state burst buffers into the storage system to absorb application I/O requests. In this paper, we simulate and explore this storage system design for use by large-scale HPC systems. First, we examine application I/O patterns on an existing large-scale HPC system to identify common burst patterns. Next, we describe enhancements to the CODES storage system simulator to enable our burst buffer simulations. These enhancements include the integration of a burst buffer model into the I/O forwarding layer of the simulator, the development of an I/O kernel description language and interpreter, the development of a suite of I/O kernels that are derived from observed I/O patterns, and fidelity improvements to the CODES models. We evaluate the I/O performance for a set of multiapplication I/O workloads and burst buffer configurations. We show that burst buffers can accelerate the application perceived throughput to the external storage system and can reduce the amount of external storage bandwidth required to meet a desired application perceived throughput goal.**

I. Introduction

High-performance computing (HPC) storage systems are designed to meet high, sustained bandwidth requirements under highly concurrent workloads. Currently, applications can achieve sustained bandwidths from 40 GiB/s to 80 GiB/s on these storage systems [18], [41]. Recent research [8], [17], [43] suggests that these storage systems are subjected to bursty I/O patterns as applications alternate between computationally dominant and I/O-dominant execution phases. In the near future, these "leadership-class" systems are expected to have 100x to 1000x more compute nodes than today's largest-scale HPC systems and require approximately 60 TiB/s of storage system bandwidth to drain application data bursts fast enough to meet checkpointing demands [12], [39].

Bursty application I/O, in conjunction with the high peak I/O rates required for future systems, creates a challenging problem for storage system designers. System designers must find a solution that allows applications to minimize time spent performing I/O (i.e., achieving a high *perceived* I/O rate) and ensure data durability in the event of failures. However, approaching this in the traditional manner—by providing a high-bandwidth external storage system—will likely result in a storage system that is underutilized much of the time. This underutilization has already been observed on current large-scale HPC systems. I/O performance data collected from the Argonne Leadership Computing Facility (ALCF) IBM Blue Gene/P system "Intrepid" indicates that the typical bandwidth to the storage system is less than one-third of its full capacity 99% of the time [8]. As the computational performance and concurrency of HPC systems continue to increase, this traditional approach leads to the deployment of increasingly expensive storage systems that are underutilized the majority of the time. An alternative, high-performance, and more economical storage system design is required to support the I/O requirements of emerging large-scale HPC systems.

Current large-scale HPC systems take advantage of enterprise parallel file systems that are designed for a range of different applications. Since customizing these file systems to meet large-scale computing needs may not be possible, a viable alternative is to augment the intermediate hardware and I/O middleware layers to better handle bursty application I/O patterns. The emergence of I/O forwarding layers [3], [27] provides a natural aggregation point and I/O offloading layer for large-scale HPC systems. This layer is low enough in the HPC I/O stack to capture all application I/O accesses and high enough in the I/O stack to complement existing storage system designs. This layer can provide a write-behind caching capability that uses NVRAM-based devices to temporarily store and manage bursts of application I/O requests.

In this paper, we explore one previously proposed [5], [7], [15] alternative storage architecture: a disk-based external storage system augmented with a tier of solid-state disk (SSD) burst buffers located in a set of I/O nodes on the periphery of

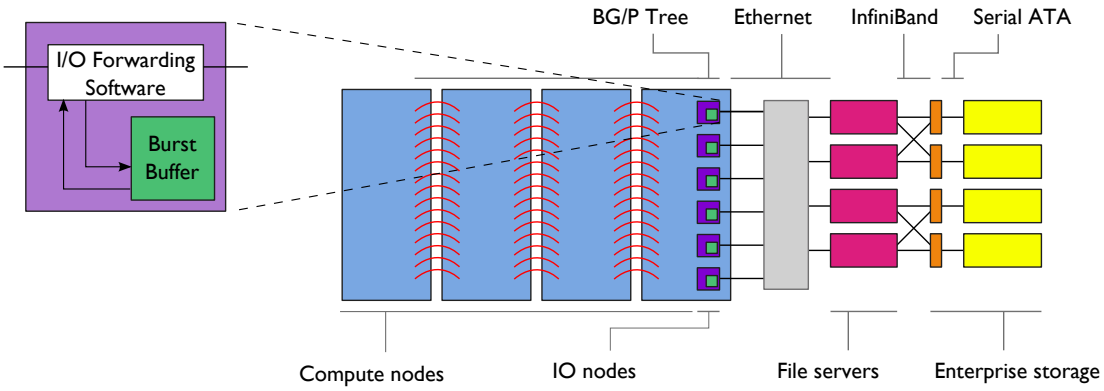


Fig. 1: Overview of the Argonne IBM Blue Gene/P (Intrepid) computing environment and storage services. This figure highlights how our proposed tier of burst buffers (green boxes) would integrate with the existing I/O nodes.

the system (Figure 1). With this tier of burst buffers, applications can push data out of memory and return to computation without waiting for data to be moved to its final resting place on an external, parallel file system. We begin (In Section II), by describing the motivating factors that lead us to investigate augmenting storage systems with burst buffers and present our design. We study this storage system architecture using the CODES parallel discrete-event storage system simulator (described in Section III). We evaluate several common I/O workloads found in scientific applications to determine the appropriate design parameters for storage systems that include burst buffers (presented in Section IV). We discuss research related to our recent work (in Section V). We conclude this paper (in Section VI) by enumerating the contributions generated by our work, in particular better quantifying the requirements of a burst buffer implementation and the degree to which external storage hardware requirements might be reduced using this approach.

II. Managing Bursty I/O

Bursty application I/O behavior is a well-known phenomenon. This behavior has been observed in prior studies for HPC applications performing periodic checkpoints [10], [19], [28], [32], [37], [43] and for the aggregate I/O activity across all applications executing within large HPC data-centers [8], [17]. To better understand the viability of the burst buffer approach, we need quantitative data on application I/O bursts so that we can accurately represent this behavior in our simulated I/O workloads. In this section, we present our analysis of bursty application I/O behavior that we observed on a large-scale HPC storage system. First, we analyze the most bursty and write-intensive applications we observed over a one-month period on a large-scale HPC system. Next, we describe how these trends hinder the performance of current systems. Then, we discuss how to manage this behavior through the use of burst buffers.

A. Study of Bursty Applications

The Argonne Leadership Computing Facility maintains the Intrepid IBM Blue Gene/P system. Intrepid is a 557 TF leadership-class computational platform and provides access to multiple petabytes of GPFS and PVFS external storage. Figure 1 provides an overview of Intrepid and the external storage services integrated with the system. Systems such as Intrepid host a diverse set of applications from many scientific domains, including climate, physics, combustion, and Earth sciences. Workloads from these scientific domains are often characterized by periodic bursts of intense write activity. These bursts result from defensive I/O strategies (e.g., checkpoints that can be used to restart calculations following a system fault) or storage of simulation output for subsequent analysis (e.g., recording time series data for use in visualization). To quantify this behavior on Intrepid, we analyzed one month of production I/O activity from December 2011 using the Darshan lightweight I/O characterization tool [9]. Darshan captures application-level access pattern information with per process and per file granularity. It then produces a summary of that information in a compact format for each job. In December 2011, Darshan instrumented approximately 52% of all production core-hours consumed on Intrepid. We identified the four most write-intensive applications for which we had complete data and analyzed the largest production example of each application. The results of this analysis are shown in Table I. Project names have been generalized to indicate the science or engineering domain of the project.

We discovered examples of production applications that generated as much as 67 TiB of data in a single execution. Two of the top four applications (Turbulence1 and AstroPhysics) illustrate the classic HPC I/O behavior in which data is written in several bursts throughout the job execution, each followed by a significant period of idle time for the I/O system. The PlasmaPhysics application diverged somewhat in that it produced only two bursts of significant write activity; the first burst was followed by an extended idle period, while the second burst occurred at the end of execution. The Turbulence2 application exhibited a series of rapid bursts that occurred nearly back-to-back at the end of execution. On a per compute node basis, the average write requests range from 0.03% to 50% of the memory size for these applications. We expect the write request per compute node to be limited by the physical

TABLE I: Top four write-intensive jobs on Intrepid, December 2011

Project	Procs	Nodes	Total Written	Run Time (hours)	Avg. Size and Subsequent Idle Time for Write Bursts >1 GiB				
					Count	Size	Size/Node	Size/ION	Idle Time (sec)
PlasmaPhysics	131,072	32,768	67.0 TiB	10.4	1	33.5 TiB	1.0 GiB	67.0 GiB	7554
					1	33.5 TiB	1.0 GiB	67.0 GiB	end of job
Turbulence1	131,072	32,768	8.9 TiB	11.5	5	128.2 GiB	4.0 MiB	256.4 MiB	70
					1	128.2 GiB	4.0 MiB	256.4 MiB	end of job
					421	19.6 GiB	627.2 KiB	39.2 MiB	70
AstroPhysics	32,768	8,096	8.8 TiB	17.7	1	550.9 GiB	68.9 MiB	4.3 GiB	end of job
					8	423.4 GiB	52.9 MiB	3.3 GiB	240
					37	131.5 GiB	16.4 MiB	1.0 GiB	322
					140	1.6 GiB	204.8 KiB	12.8 MiB	318
Turbulence2	4,096	4,096	5.1 TiB	11.6	21	235.8 GiB	59.0 MiB	3.7 GiB	1.2
					1	235.8 GiB	59.0 MiB	3.7 GiB	end of job

memory of the node (2 GiB on Intrepid). From the I/O node perspective, the write burst sizes range from 40 MiB to 67 GiB (or 0.0511% to 52.3% of the total amount of memory available to the application). Also, the observed idle time between two write bursts varies among the applications from a couple of minutes to as long as two hours. They all feature an end-of-job write burst. We used these statistics to guide the parameterization of our simulated application I/O workloads.

B. Impact on Storage Systems

External storage systems, such as the one integrated with Intrepid, are designed to quickly and efficiently store application checkpoint data. This design point leads to storage systems that may be idle during application computation phases and saturated during application I/O phases. We have observed extreme cases of bursty application I/O patterns and the lack of storage system utilization in prior work [8]. We made several observations over a two-month period, that indicate that the achievable throughput to Intrepid's external storage system is often not realized, except for sporadic bursts of I/O activity that correspond with storage system high utilization. In prior work, Intrepid's storage bandwidth was observed to be at 33% or less utilization for 99.2% of the time over a two-month period. Over this same period, Intrepid's external storage system operated at 5% or less of its expected peak bandwidth for 69.2% of the time. While the maximum observed throughput over this interval was 35.05 GiB/s, the average throughput was 1.93 GiB/s. These observations indicate that a lower-bandwidth external storage system could provide the required level of service, if bursty I/O traffic could be spread out over a longer period of time (while allowing applications to resume computation). Given the decreasing cost of solid-state storage, this approach is likely to be more economical.

C. Absorbing Bursty I/O Patterns

One solution to handling I/O bursts in large-scale HPC systems is to absorb the I/O bursts at an intermediate storage layer consisting of burst buffers. Burst buffers are high-throughput, low-capacity storage devices that act as a staging area or a write-behind cache for HPC storage systems. A compelling approach to incorporating burst buffers is to place these buffers on I/O nodes that connect to the external storage system and to manage these buffers as part of the I/O forwarding services. Figure 1 illustrates how burst buffers could be integrated within Intrepid's existing infrastructure. We envision that burst buffers will integrate at HPC I/O nodes and will be managed by I/O forwarding software. If the burst buffers are sufficiently large and fast, they can absorb the relatively infrequent I/O bursts observed during our studies.

From the data we gathered during our recent workload study, we see that three of the top four production applications on Intrepid could benefit greatly from a burst buffer architecture in terms of potential reduction in perceived I/O time. The majority of existing applications exhibit several periods of idle I/O activity between I/O bursts. By aggregating and absorbing the I/O requests into the burst buffer layer, applications can overlap computations that follow I/O bursts while bleeding the data from the burst buffer to external storage. Without these burst buffers, applications would block until all I/O requests had completed and would allow no potential for optimization or overlapping computation and I/O activity. The Turbulence2 application would likely benefit the least because all of its data is written at the end of execution, allowing no opportunity to overlap buffer flushing activity with application computation. However, if data were allowed to trickle out of the burst buffer after the job was complete, the Turbulence2 application would still see a reduction in overall runtime.

III. Modeling HPC Storage Systems and Applications

Our simulation tools are built on top of the Rensselaer Optimistic Simulation System (ROSS). Using ROSS, we have implemented and validated [20] a storage system simulator for the CODES exascale storage system project. This simulator models the storage system hardware and software protocols used by the ALCF's Intrepid IBM Blue Gene/P system. As part of this work, we extended this storage system simulator to include a burst buffer tier of storage and updated the I/O forwarding software protocols to manage data stored in the burst buffers. In the remainder of this section, we present our tools used in the simulation component of our burst buffer study.

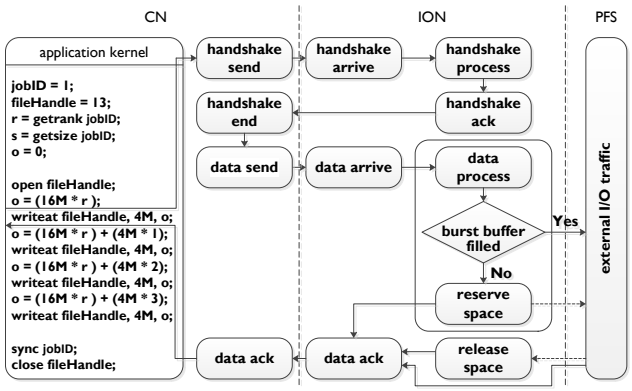


Fig. 2: Integration of burst buffer model into the CODES storage system write request model; dotted arrows represent asynchronous I/O when burst buffer is in use.

A. Parallel Discrete-Event Simulation

ROSS is a massively parallel discrete-event simulator that has demonstrated the ability to process billions of events per second by leveraging large-scale HPC systems [4], [21]. A parallel discrete-event simulation (PDES) system consists of a collection of *logical processes*, or LPs, each modeling a distinct component of the system being modeled (e.g., a file server). LPs communicate by exchanging timestamped event messages (e.g., denoting the arrival of a new I/O request at that server). The goal of PDES is to efficiently process all events in a global timestamp order while minimizing any processor synchronization overheads. Two well-established approaches toward this goal are broadly called conservative processing and optimistic processing. ROSS supports both approaches. All results presented in this paper use the conservative approach.

B. CODES Storage System Model

The end-to-end storage system model is composed of several component models that capture the interactions of the system software and hardware for I/O operations. The models used in our simulations include networks, hardware devices, and software protocols. The storage system model also provides several configuration parameters that dictate the execution behavior of application I/O requests.

We abstracted the common features of each Blue Gene/P hardware component into compute node (CN), I/O node (ION), and parallel file system (PFS) models. The PFS model includes file server and enterprise storage submodels. These models are the logical processes in our end-to-end storage system model, which are the most basic units of our parallel discrete-event model. The various BG/P networks are modeled as the links connecting each LP. Each LP includes three buffers. The incoming buffer is used to model the queuing effects from multiple LPs trying to send messages to the same LP. The outgoing buffer is used to model queuing effects when an LP tries to send multiple messages to different LPs. The processing buffer is used to model queuing effects caused by a processing unit, such as CPU, DMA engine, storage controller, or router processors. The units process incoming messages in FIFO order.

The network connection between two LPs is modeled as messages transmitted between the two LPs, where each LP's incoming buffer is connected to the other LP's outgoing buffer. Furthermore, the commodity networks (Ethernet and Myrinet networks) are modeled by the links connecting the IONs with the storage servers. If we increase the fidelity of our models in the future, additional network components, such as routers and switches, can be modeled as LPs.

In prior work, we evaluated the accuracy of these models [20]. The storage system models were developed to be software protocol-level accurate. We opted for protocol-level fidelity over cycle-level fidelity to provide a simulation framework that is high-performance and accurate enough for our experiments. We validated these models using data collected on Argonne's Intrepid Blue Gene/P system while Intrepid was being deployed. Our simulated results accurately reflected the performance variations reported in prior work [18] from 2,048 to 131,072 client processes, at roughly 10% error rate.

C. Burst Buffer Support in the CODES Storage System Model

We made several modifications to the existing CODES storage system simulator to support a tier of burst buffer storage. First, we developed a simple model of a solid-state storage device, identified the parameters of interest for modeling these devices, and consulted the literature for current SSD products to define reasonable values for these parameters. Next, we updated the I/O node hardware model in the CODES simulator to support the integration of burst buffers. Then, we updated the write request protocol of our simulator so that the I/O forwarding software layer managed I/O requests stored in the burst buffers.

$$T = \frac{L_{data}}{B_{BB}} + T_{BB} \tag{1}$$

Equation 1 describes the analytical model used by our simulator to compute the burst buffer data access costs. We define the access time (T) such that it is influenced by the data transfer size (L_{data}), the device throughput (B_{BB}), and the data access latency (T_{BB}). This is a naïve model of SSD devices; it assumes the same costs for read and write operations, and it ignores decreases in device reliability and write endurance [6], [33]. However, this model is sufficient for approximating the general performance profile of an end-to-end storage system using these devices since the fidelity of this model is on a par with other hardware models used by our simulator.

To identify realistic parameters for the burst buffer device model, we investigated the parameters and characteristics of several solid-state storage devices. Currently, several solid-state storage devices are appropriate for use as burst buffers. Table II summarizes the capacity, latency, and throughput parameters for some of these devices. In general, these devices provide between 0.25 TiB and 1.4 TiB of storage capacity, write throughputs ranging from 0.21 GiB/s to 1.3 GiB/s, and access latencies between 15 µs and 80 µs, which are on a par

TABLE II: Summary of relevant SSD device parameters and technology available as of January 2012.

	Size		Bandwidth (GiB/s)		Latency (µs)	
Vendor	(TiB)	NAND	Write	Read	Write	Read
FusionIO	0.40	SLC	1.30	1.40	15	47
FusionIO	1.20	MLC	1.20	1.30	15	68
Intel	0.25	MLC	0.32	0.50	80	65
Virident	0.30	SLC	1.10	1.40	16	47
Virident	1.40	MLC	0.60	1.30	19	62

with practical application requirements described in Table I. On Intrepid the compute node to I/O node ratio is 64:1, with each compute node having 2 GiB of main memory. The smallest solid-state device in our survey has a capacity of 0.25 TiB, which is more than enough to hold all the data in main memory on all associated compute nodes in a single burst. We used the range of device capacity, access latency, and throughput parameters to dictate possible storage system configurations in our simulations.

Figure 2 illustrates how the burst buffer model integrates with existing CODES storage models. At the compute node level, the write request kernel is translated to a simulator trigger event, and the control is passed to the underlying simulation engine. All the following events (boxes) represent the model details of the protocols used in the storage system. In the original protocol, the compute nodes forward application I/O requests to the I/O nodes, and the I/O forwarding software replays these requests to the file system. To support burst buffer storage at the I/O nodes, we modified the I/O forwarding write request protocol to manage application data cached in these devices. Our burst buffer data management protocol model is similar to a write-behind cache policy. First, I/O forwarding software attempts to reserve space in the burst buffer for the application data. The I/O forwarding software then receives the application data and deposits this data into the burst buffer. Once the data is buffered, the I/O forwarding software signals the application that the write operation completed. The I/O forwarding software then transfers the buffered data to the file system. No other adjustments were made to the write request protocol. The application client can force all data in the burst buffer to be flushed to the file system through a commit method. The details of the file servers and disk level model are not addressed in Figure 2. Additional details of these models are documented in our prior work [20].

Several parameters can influence application I/O behavior in this model. The solid-state device parameters include latency, bandwidth, and capacity. Latency determines the data access rates to these devices, and capacity is the amount of buffer space that the devices can provide. The memory copy bandwidth on the I/O nodes also limits the rate at which the application data payload can be transferred between RAM and the burst buffers. The storage network throughput and the disk-based storage system throughput dictate how quickly the burst buffers can be flushed. The compute node network controls how quickly we can fill the burst buffers with application data. If sufficient space is available to buffer the application data, applications will transfer data directly to the burst buffer and avoid other storage system protocol or hardware costs. However, the cost of flushing the burst buffers and the protocol interactions with the external storage system are still accounted for and visible to the applications. If space is not available to buffer the application data, the original write request protocol (without burst buffer support) is executed.

D. CODES I/O Workloads

To drive the simulator, we used several workloads derived from the I/O patterns of synthetic benchmarks and scientific applications. In our prior work [20], the I/O workloads were tightly integrated with our simulator. To further generalize our storage simulator and to allow us to evaluate multiple types of I/O workloads, we developed a small I/O description language and interpreter. We integrated the language interpreter into the CODES simulator. The language and interpreter allow us to describe a variety of application workloads; provide features to dictate the placement of applications within the system; and define I/O workloads consisting of multiple, parallel, concurrently executing applications. For the experiments we present in this paper, we developed several I/O kernels based on the IOR synthetic benchmark, the FLASH astrophysics application, and general I/O kernels that mimic the I/O patterns described in Table I.

The first I/O kernel we developed was for the IOR benchmark. IOR is a suitable proxy for some HPC I/O workloads and is often used for evaluating supercomputer I/O performance [40]. The IOR I/O pattern represented in our I/O kernel consists of many processes concurrently writing large blocks of data (multiple megabyte chunks) into a shared file. We validated the results generated by this kernel against results generated during our prior work [20].

Next, we evaluated the I/O workload for the FLASH astrophysics code [36] and developed an I/O kernel for this code. FLASH is a scientific application used to study nuclear flashes that occur on the surfaces of white dwarfs and neutron stars. It is an extremely scalable scientific application that successfully reached scales of at least 65,536 cores on Argonne's Intrepid Blue Gene/P system [19]. For our experiments, we distilled two phases of the FLASH I/O workload: the checkpoint I/O phase and the plot file I/O phase.

During the checkpoint I/O phase of FLASH, each process interleaves several large blocks of variable data stored in double-precision floating-point format into a single, shared data file. During the plot file I/O phase, each process interleaves several medium sized blocks of variable data stored in single-precision floating-point format into a single, shared data file. FLASH writes data for each variable as a contiguous buffer into the checkpoint and plot files. The high-level I/O library used and I/O optimizations enabled by this library can dictate the I/O pattern used to describe the data generated by the application. For example, using the HDF5 high-level I/O library with collective I/O optimizations disabled will force all application processes to write several small blocks (between

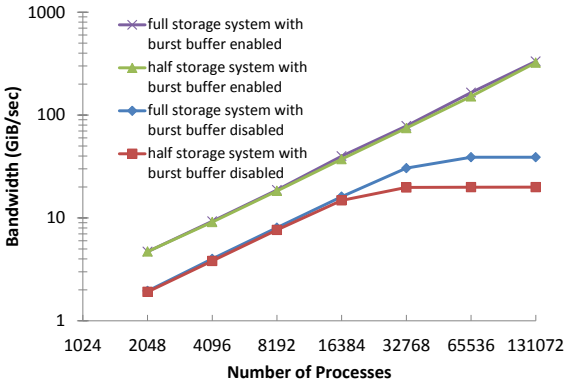


Fig. 3: Simulated performance of IOR for various storage system and burst buffer configurations.

100 and 4,000 bytes) to the file before writing the larger variable data blocks. Many of the file accesses are unaligned.

We evaluated the FLASH I/O kernel at a scale of 65,536 application processes in our storage system simulator. Our I/O kernel was configured to mimic the HDF5 I/O behavior of FLASH when collective I/O optimizations are disabled. In prior work [18], we observed on Intrepid that FLASH checkpoint files could be stored at 21.05 GiB/s and the FLASH plot file could be stored at 6.4 GiB/s. Initially, we observed that the simulator stored the FLASH checkpoint file data at 30.77 GiB/s and the plot file data at 15.51 GiB/s. We discovered that the simulator did not correctly account for the saturation of the commodity storage network and did not correctly handle small HDF5 I/O requests (100s to 1000s of bytes) written to the enterprise storage model. We adjusted our commodity network model to account for network saturation when using more than 65,536 processes and penalized small I/O requests written to the enterprise storage device model. With these changes, the simulator computed the checkpoint file data storage rate as 20.01 GiB/s and the plot file data storage rate as 6.715 GiB/s.

IV. Burst Buffer Study

In this section, we explore the burst buffer storage system design using our simulator. First, we investigate how burst buffers influence individual application I/O behavior. Next, we explore how burst buffers influence the performance and design of the external storage system when simulating multiple, concurrent applications.

A. Single I/O Workload Case Studies

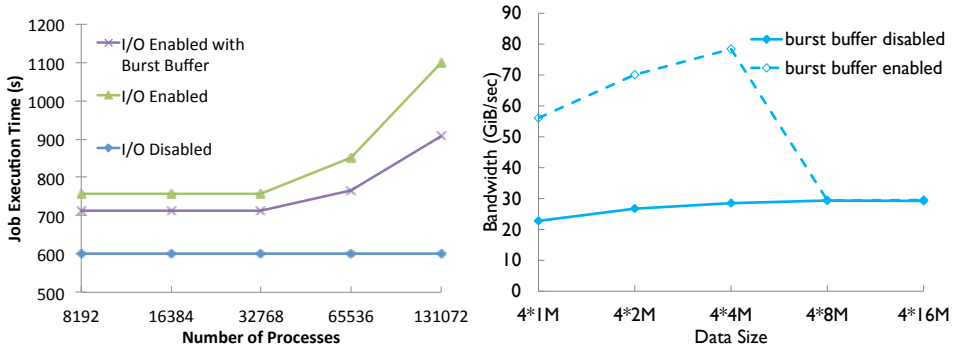
To understand how burst buffers influence application I/O performance, we evaluated several I/O workloads in our simulator. In these experiments, we focused on exploring the parameter space of the burst buffers and application I/O access patterns. We limited these analyses to a single application I/O workload so that we could observe the impact of burst buffers from the application's perspective.

The goal of our first experiment was to quantify the I/O acceleration that burst buffers can provide to an application. These experiments used the IOR I/O kernel we described in Section III-D. We configured the I/O kernel in these experiments to write four consecutive chunks of data 4 MiB in length. We configured the simulated storage system to use 4 MiB stripes. Thus, the I/O kernel will write stripe-aligned data requests to the storage system and should achieve the best possible throughput for the I/O kernel. The configuration of the simulated storage system was similar to experiments presented in our prior work [20]. This configuration included 123 file servers and 123 enterprise storage LUNs connected to the Blue Gene/P system through a commodity storage network. File system clients were hosted on the I/O nodes and groups of 256 compute processes (on 64 compute nodes) shared access to a single I/O node.

In addition to these configuration parameters, we made two adjustments to the experimental setup of our simulation. First, we configured an additional external storage system that consists of 64 file servers and 64 enterprise storage LUNs. This additional storage system configuration is approximately half the storage system provided by the existing ALCF computing environment. Next, we configured each I/O node to use a 4 GiB burst buffer with a transfer rate of 1.5 GiB/s. While a 4 GiB burst buffer is substantially smaller than the devices presented in Table II, this size can fit all the application data generated by this experiment. Increasing the burst buffer size does not affect the performance of the model or the simulation.

Figure 3 illustrates the perceived I/O bandwidth at the client processes for a variety of scales. This bandwidth accounts for the time for the applications to complete its I/O requests. It does not account for the time to open or close the file, or the time to ensure the durability of the application data on the external storage system. Thus, the results of this experiment illustrate the maximum achievable application I/O bandwidth for storage systems with and without burst buffers. The full storage system without a burst buffer achieves sustained bandwidth similar to our prior study [20] because all application I/O requests interact with the external storage system. The half storage system configuration without a burst buffer exhibits similar I/O performance trends but achieves only half the bandwidth of the full storage system. When burst buffers are enabled for either external storage system configuration, we achieve linear scaling of I/O bandwidth. The burst buffers essentially cache the application I/O requests and suppress the cost of interacting with the external storage system. In this test, the application I/O performance is limited by the bandwidth of the Blue Gene/P tree network that connects the compute nodes with the I/O nodes. This network has a maximum bandwidth of 700 MiB/s for 4 MiB I/O requests.

In order to observe the benefits of the burst buffers when accounting for data flushes, applications must overlap computations with the burst buffer flush. We modified our initial IOR experiment to account for the cost of flushing burst buffer data so that we could quantify the benefit of computation and I/O overlap. This new experiment measures application runtimes at various scales for storage systems configured with and without burst buffers. In this experiment, we used the full



(a) Simulated application runtime performance for various I/O and burst buffer configurations.

(b) Simulated IOR performance for various burst buffer configurations and I/O workloads.

Fig. 4: Results of the burst buffer and application I/O workload parameter space investigations.

storage system configuration consisting of 123 file servers. We added a 20 GiB burst buffer to each I/O node.

Figure 4a illustrates the simulated application runtime (computation time plus I/O time) at various scales. We collected data for three application configurations. Each application executed two computation phases that consumed five minutes of application runtime. First, we measured the application runtime when the application executed no I/O operations. This experiment represents the best possible case for the application because it does not interact with the storage system and is not affected by external storage system costs. Next, we measured the runtime for an application that performs I/O directly to the external storage system and does not use a burst buffer. After each computation phase, each process enters an I/O phase and writes four 20 MiB chunks of data. For this application, this test case represents the worst possible case for the application runtime, since the I/O is not overlapped with application execution. Then, we measured the application runtime with burst buffers enabled. Similar to the previous test case, each process writes four 20 MiB chunks of data after each computation phase. However, the I/O forwarding layer completes the application I/O request once the data is transferred to the burst buffer. Thus, the application computation can overlap the burst buffer data transfers to the external storage system. This test case highlights how the application runtime decreases when the application computation is overlapped with the I/O forwarding layer writing burst buffer data to external storage.

Next, we investigated how the application bandwidth is affected by the burst buffer capacity and application I/O request sizes. The results of this experiment are illustrated in Figure 4b. For this experiment, we configured the IOR I/O kernel to issue four I/O requests ranging from 1 MiB to 16 MiB. We evaluated this I/O pattern using 32,768 IOR processes, 123 file servers, and a 4 GiB burst buffer on each I/O node. We measured the perceived application I/O bandwidth and ignored the cost of flushing the burst buffer data to the external storage system.

When the burst buffer is disabled, the application I/O performance is limited by the external storage system performance for all I/O request sizes. When the burst buffer is enabled, the application I/O performance is limited by the time to transfer data from the compute node into the burst buffer while the burst buffer still has free space. In this experiment, this occurs at 4 MiB request size test. After this point, the application I/O performance is limited by the external storage system because the I/O requests overflow the burst buffer.

B. Multiapplication Case Study

An interesting question that arises when burst buffers are introduced into the system is the degree to which multiple applications running simultaneously might conflict with one another and in what ways. I/O bursts that saturate the storage system can delay or starve data accesses generated by other applications competing for access to the same storage resources [23]. Interleaved I/O requests from multiple, unrelated sources can lead to random I/O workloads for storage devices to handle and often require additional software to effectively reorder these requests [24], [30]. As part of our study we simulated the behavior of the system with multiple applications. We note that the current I/O forwarding and burst buffer models do not perform any intelligent reordering or transformation on the I/O request stream; rather, they replay operations in order on each I/O forwarder.

For this study we initialized the simulator to have 32K compute nodes and 512 IONs. The SLC NAND FusionIO card parameters were used for the burst buffer model (i.e., 400 GiB capacity, 1.3 GiB/sec write rate). Two external storage configurations were tested: 128 file servers with 16 enterprise storage racks (full storage system) and 64 file servers with 8 enterprise storage racks (half storage system). We separated the compute nodes into three partitions: two 8K node partitions and one 16K node partition.

From the data in Table I we generated three application I/O patterns that reflect the patterns seen in the PlasmaPhysics, AstroPhysics, and Turbulence1 applications. The PlasmaPhysics kernel was configured so that each application process issued two large (256 MiB) write request bursts separated by a two-hour interval. The AstroPhysics kernel was set up to

execute three small I/O phases followed by a large I/O phase, where each phase was separated by a five-minute interval. This pattern was repeated eleven times. The Turbulence1 I/O kernel executed 220 small I/O phases separated by a 70-second interval. We configured the simulation to run for 5 simulated hours, enough time for the "applications" to reach steady state and for us to observe I/O activities and conflicts.

We simulated test cases with burst buffers enabled and disabled. Additionally, we conducted tests using the full and half external storage system configurations. The results of these simulations are illustrated in Figures 5 and 6. During these simulations, we collected 2,000 samples of the total amount of data transferred throughout the simulator at ten second intervals. The ten second average data transfer rates are reported in Figures 5 and 6.

One of the observations we made from the multiapplication experiment is that burst buffers accelerate the application perceived throughput under mixed I/O workloads. The 400 GiB burst buffer was large enough to buffer the data requests generated by all three workloads. Additionally, decreasing the size of the storage system while using burst buffers had no noticeable impact on the mixed I/O workloads performance. Without burst buffers, decreasing the external storage system by half its original size impacted the applications' I/O performance. Figures 5a and 5b show that the peak bandwidth of the storage system is slightly less in the half storage system configuration, and the time to complete all the I/O requests is extended. When burst buffers are enabled, decreasing the external storage system still provides exceptional I/O bandwidth for the applications. Figures 5c and 5d indicate no observable difference in aggregate I/O performance when the storage system is decreased by half its original size.

Figures 6a through 6d illustrate simulated disk I/O statistics monitored during same time period. When the burst buffer is enabled, the disk I/O requests appear more condensed resulting in better disk utilization. Comparing Figure 6c and 6d, we find that job execution time is approximately the same, which shows an even higher disk utilization in the half storage system test case. The burst buffer is capable of feeding the storage system with large enough I/O requests, and these requests are fully digested by storage system, in the time intervals between the write requests. This indicates the potential of saving storage resources with the burst buffer approach while hitting performance targets.

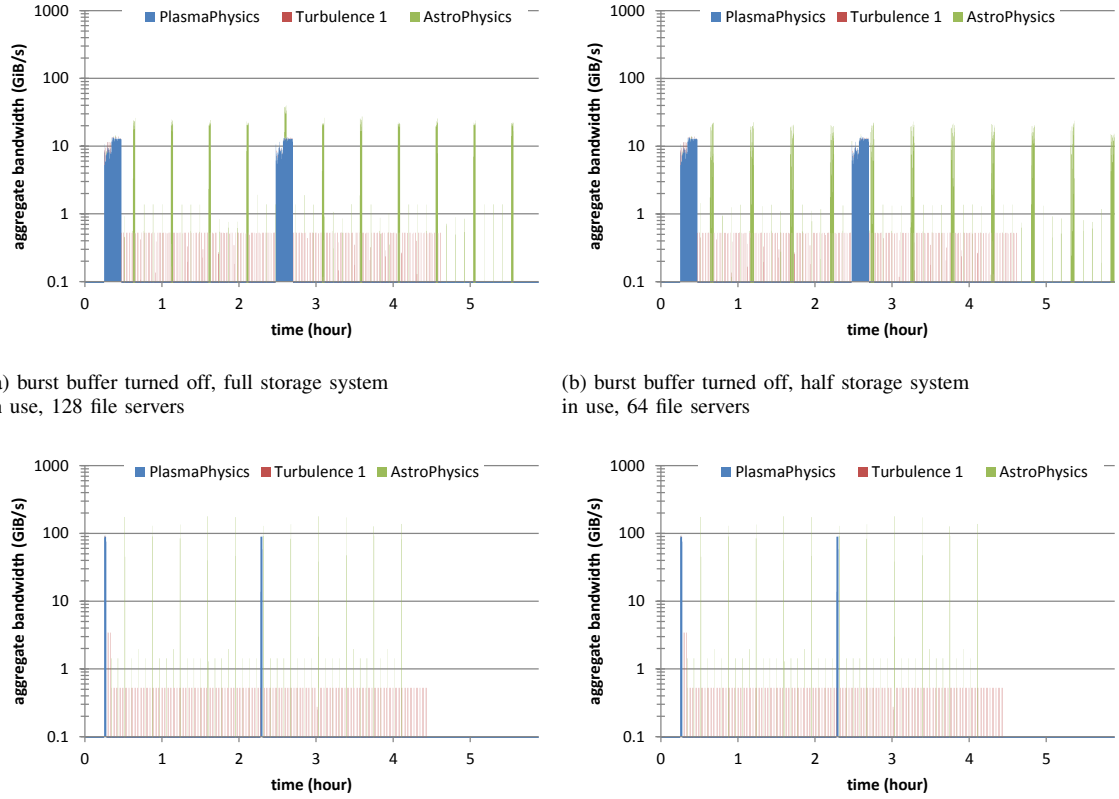
V. Related Work

Three areas of work are related to our research. First, a significant amount of research has been devoted to accurate and scalable simulation methods and frameworks for HPC systems, with a relative increase in activity to generate simulation tools for designing exascale computing systems. The second area of related research involves asynchronous I/O methods for HPC systems; this area focuses on identifying techniques for processing HPC application I/O requests in the background of application execution. The third area of related work focuses on integrating NVRAM or solid-state storage into HPC systems for use as a fast tier of storage.

As part of the exascale co-design process, significant interest has arisen in understanding how parallel system software such as MPI and the associated supercomputing applications will scale on future architectures. For example, Perumalla's $\mu\pi$ system [34] will allow MPI programs to be transparently executed on top of the MPI modeling layer and simulate the MPI messages. A number of universities and national laboratories have joined together to create the Structural Simulation Toolkit (SST) [35]. SST includes a collection of hardware component models including processors, memory, and networks at different accuracy. These models use parallel, component-based discrete-event simulation based on MPI. BigSim [44] focuses on modeling and predicting the behavior of sequential execution blocks of large-scale parallel applications. Our simulator differs from these projects because our tools support the construction of storage system models and address how to represent the storage system software protocols in parallel discrete-event simulators.

Researchers have also developed a number of parallel file system simulators. The IMPIOUS simulator [26] was developed for fast evaluation of parallel file system designs. It simulates PVFS, PanFS, and Ceph file systems based on user-provided file system specifications, including data placement strategies, replication strategies, locking disciplines, and caching strategies. The HECIOS simulator [38] is an OMNeT++ simulator of PVFS; it was used to evaluate scalable metadata operations and file-data-caching strategies for PVFS. PFSsim [22] is an OMNeT++ PVFS simulator that allows researchers to explore I/O scheduling algorithm design. PVFS and ext3 file systems have been simulated by using colored Petri nets [29]; this simulation method yielded low simulation error, with less than 10% error reported for some simulations. Checkpoint workloads and storage system configurations were recently evaluated with the SIMCAN simulator, an OMNeT++ simulator for HPC architectures [31]; from the SIMCAN simulations, the authors concluded that increasing the performance of the external storage system and storage networks were effective methods for improving application checkpoint performance. The focus of CODES sets it apart from these related simulation tools. One of the goals of CODES is to accurately and quickly simulate large-scale storage systems. To date, CODES has been used to simulate up to 131,072 application processes, 512 PVFS file system clients, and 123 PVFS file servers. The existing studies limited their simulations to smaller parallel systems (up to 10,000 application processes and up to 100 file servers).

An abundance of research focus on asynchronous file I/O, data staging, and I/O offloading research for HPC systems. Many of the recent challenges associated with this research area focus on how to minimize the impact of asynchronous file I/O network activity on application communication over shared interconnects. GLEAN [42] and DART [11] provide data offloading and staging capabilities for use by HPC applications in data-center and wide-area computing environments. Both of these tools ship application I/O requests to data staging



(a) burst buffer turned off, full storage system in use, 128 file servers

(b) burst buffer turned off, half storage system in use, 64 file servers

(c) burst buffer turned on, full storage system in use, 128 file servers

(d) burst buffer turned on, half storage system in use, 64 file servers

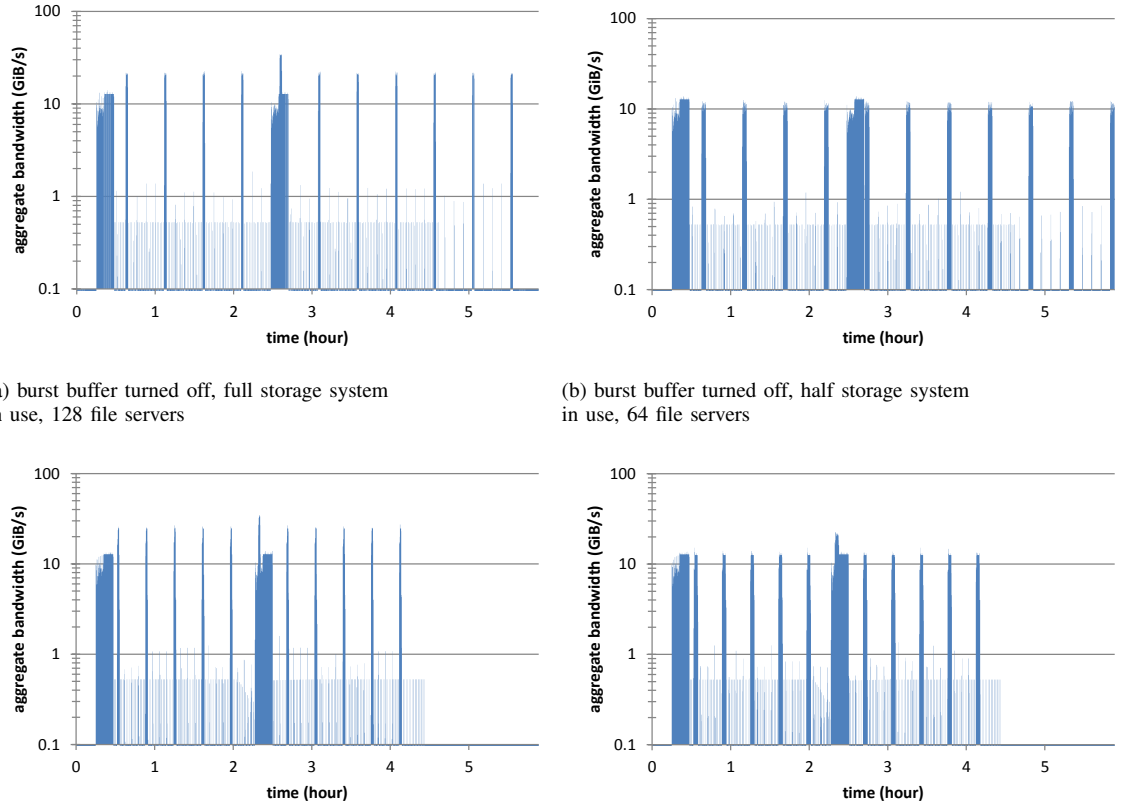
Fig. 5: Ten second average data transfer rate for the compute nodes observed during the multiapplication simulations.

nodes and allow these nodes to manage the I/O requests on behalf of the application. The ADIOS and DataStager teams have done extensive research on how to adapt and isolate applications from several sources of interference, including sources that impact asynchronous I/O capabilities [1] and usage of heavily utilized file system resources [23]. Our work is complementary to that existing work. Those asynchronous tools provide the interfaces and mechanisms to provide asynchronous I/O capabilities to the application, whereas our work provides insight on how those tools will work on large-scale systems with a dedicated buffer space.

Several recent activities have focused on the usage of solid-state devices in HPC systems. This work includes evaluating the use of solid-state storage for data-intensive computations [13], [25], investigating the use of solid-state devices in storage systems [2], designing data-intensive HPC systems that can host large data sets near processing elements [16], and using SSDs as a medium for temporarily storing application checkpoint data [14], [30]. Of particular interest and relevance to our work are storage system designs that use nonvolatile burst buffers [5]. These system designs integrate nonvolatile media between the main memory accessible to the processing elements and the slower spinning disk media that hold durable copies of application data. Thus, it is used as an intermediate storage tier located between the processing elements and the external storage system. In this paper, we investigate the role of burst buffers in large-scale HPC systems. We adopt the usage model of nonvolatile burst buffers [5], define several large-scale HPC storage system configurations integrated with burst buffers, and evaluate application I/O workloads using these system configurations through the CODES storage system simulator.

VI. Conclusions

This study explores the potential for burst buffers in HPC storage systems and highlights their potential impact through simulations of an existing large-scale HPC system, the Argonne IBM Blue Gene/P "Intrepid" including application I/O patterns and enhanced storage system. We gather data from the production computing system to provide an accurate model of application I/O bursts in today's systems, and we survey current nonvolatile storage products to provide realistic parameters for our burst buffer model. We provide a parallel discrete-event model for burst buffers in an HPC system based on our previous work of a leadership-class storage system model. The model is tested through a series of I/O benchmarks and mixed workloads, and simulations are performed with a reduced configuration of external storage.



(a) burst buffer turned off, full storage system in use, 128 file servers

(b) burst buffer turned off, half storage system in use, 64 file servers

(c) burst buffer turned on, full storage system in use, 128 file servers

(d) burst buffer turned on, half storage system in use, 64 file servers

Fig. 6: Ten second average data transfer rate for the external storage system observed during the multiapplication simulations.

This study clearly indicates that burst buffers are practical in the context of this example system and set of applications. While current systems operate effectively without burst buffers, we show that the use of burst buffers can allow for a much less capable external storage system with no major impact on perceived application I/O rates. For today's systems this means that storage system costs could likely be significantly reduced—fewer file servers, racks of storage, and external switch ports are needed. For systems in the 2020 time frame, burst buffers are likely to be a mandatory component if peak I/O rates are to be attained.

Future work will follow along two lines. First, in order to further improve the efficiency of our simulations on large-scale systems, we will adapt our models to optimistic execution. Second, we continue to improve the fidelity of the model. We are incorporating models of the torus and tree networks as part of this activity, and we are investigating methods to better simulate the disk drives in our enterprise storage model.

ACKNOWLEDGMENTS

This work was supported in part by the Office of Advanced Scientific Computer Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357 and partially by Contract DE-SC0005428 and the LANL/UCSC Institute for Scalable Scientific Data Management (ISSDM). 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.

REFERENCES

[1] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. DataStager: Scalable data staging services for petascale applications. In *Proceedings of the 18th ACM international Symposium on High Performance Distributed Computing*, June 2009.

[2] S. Alam, H. El-Harake, K. Howard, N. Stringfellow, and F. Verzelloni. Parallel I/O and the metadata wall. In *Proceedings of the 6th Parallel Data Storage Workshop (PDSW'11)*, November 2011.

[3] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan. Scalable I/O forwarding framework for high-performance computing systems. In *IEEE International Conference on Cluster Computing (Cluster 2009)*, New Orleans, LA, September 2009.

[4] D. W. Bauer Jr., C. D. Carothers, and A. Holder. Scalable time warp on Blue Gene supercomputers. In *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, pages 35–44, Washington, DC, USA, 2009. IEEE Computer Society.

[5] J. Bent and G. Grider. Usability at Los Alamos National Lab. In *The 5th DOE Workshop on HPC Best Practices: File Systems and Archives*, September 2011.

[6] S. Boboila and P. Desnoyers. Write endurance in flash drives: Measurements and analysis. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, February 2010.

[7] D. L. Brown and P. Messina. Scientific grand challenges: Cross-cutting technologies for computing at the exascale. Technical report, Department of Energy Office of Advanced Scientific Computing Research and Office of Advanced Simulation and Computing, February 2010.

[8] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross. Understanding and improving computational science storage access through continuous characterization. *Trans. Storage*, 7:1–26, October 2011.

[9] Philip Carns, Robert Latham, Robert Ross, Kamil Iskra, 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 (IASDS)*, New Orleans, LA, September 2009.

[10] J. Dennis and R. Loft. Optimizing high-resolution climate variability experiments on the Cray XT4 and XT5 systems at NICS and NERSC. In *Proceedings of the 51st Cray User Group Conference (CUG)*, 2009.

[11] C. Docan, M. Parashar, and S. Klasky. Enabling high-speed asynchronous data extraction and transfer using DART. *Concurrency and Computation: Practice and Experience*, 22(9):1181–1204, June 2010.

[12] J. Dongarra. Impact of architecture and technology for extreme scale on software and algorithm design. Presented at the Department of Energy Workshop on Cross-cutting Technologies for Computing at the Exascale, February 2010.

[13] B. Van Essen, R. Pearce, S. Ames, and M. Gokhale. On the role of NVRAM in data-intensive architectures: an evaluation. In *International Symposium on Parallel and Distributed Processing (to appear)*, 2012.

[14] L. Gomez, M. Maruyama, F. Cappello, and S. Matsuoka. Distributed diskless checkpoint for large scale systems. In *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud, and Grid Computing (CCGrid'10)*, May 2010.

[15] G. Grider. Exa-scale FSIO - Can we get there? Can we afford to? Presented at the 7th IEEE International Workshop on Storage Network Architecture and Parallel I/O, May 2011.

[16] J. He, J. Bennett, and A. Snavely. DASH-IO: and empirical study of flash-based IO for HPC. In *Proceedings of TeraGrid'10*, August 2010.

[17] Y. Kim, R. Gunasekarana, G. Shipman, D. Dillow, Z. Zhang, and B. Settlemyer. Workload characterization of a leadership class storage cluster. In *Proceedings of the 5th Parallel Data Storage Workshop (PDSW'10)*, November 2010.

[18] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock. I/O performance challenges at leadership scale. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, page 40. ACM, 2009.

[19] R. Latham, C. Daley, W. K. Liao, K. Gao, R. Ross, A. Dubey, and A. Choudhary. A case study for scientific I/O: Improving the FLASH astrophysics code. In *under review to Journal of Computational Science and Discovery*, 2012.

[20] N. Liu, C. Carothers, J. Cope, P. Carns, R. Ross, A. Crume, and C. Maltzahn. Modeling a leadership-scale storage system. In *Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics*, 2011.

[21] N. Liu and C. D. Carothers. Modeling billion-node torus networks using massively parallel discrete-event simulation. In *Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation*, PADS '11, pages 1–8, Washington, DC, USA, 2011. IEEE Computer Society.

[22] Y. Liu, R. Figueiredo, D. Clavijo, Y. Xu, and M. Zhao. Towards simulation of parallel file system scheduling algorithms with PFSsim. In *Proceedings of the 7th IEEE International Workshop on Storage Network Architectures and Parallel I/O*, May 2011.

[23] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf. Managing variability in the IO performance of petascale storage systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis 2010 (SC10)*, November 2010.

[24] T. Madhyastha, G. Gibson, and C. Faloutsos. Informed prefetching of collective input/output requests. In *Proceedings of the 1999 ACM/IEEE Conference on Supercomputing (SC'99)*, November 1999.

[25] N. Master, M. Andrews, J. Hick, S. Canon, and N. Wright. Performance analysis of commodity and enterprise class flash devices. In *Proceedings of the 5th Parallel Data Storage Workshop (PDSW'10)*, November 2010.

[26] E. Molina-Estolano, C. Maltzahn, J. Bent, and S. Brandt. Building a parallel file system simulator. In *Journal of Physics: Conference Series*, volume 180, 2009.

[27] J. Moreira, M. Brutman, J. Castaños, T. Engelsiepen, M. Giampapa, T. Gooding, R. Haskin, T. Inglett, D. Lieber, P. McCarthy, M. Mundy, J. Parker, and B. Wallenfelt. Designing a highly-scalable operating system: the Blue Gene/L story. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, November 2006.

[28] H. Naik, R. Gupta, and P. Beckman. Analyzing checkpointing trends for applications on petascale systems. In *Second International Workshop on Parallel Programming Models and Systems Software (P2S2) for High-End Computing*, 2009.

[29] H. Q. Nguyen. *File system simulation: Hierachical performance measurement and modeling*. PhD thesis, University of Arkansas, 2011.

[30] P. Nowoczynski, N. Stone, J. Yanovich, and J. Sommerfield. Zest: Checkpoint storage system for large supercomputers. In *3rd Petascale Data Storage Workshop*, November 2008.

[31] A. Nunez, J. Fernandez, J. Carretero, L. Prada, and M. Blaum. Optimizing distributed architectures to improve performance on checkpointing applications. In *Proceedings of the 13th IEEE International Conference on High Performance Computing and Communications (HPCC'11)*, September 2011.

[32] R. Oldfield, S. Arunagiri, P. Teller, S. Seelam, M. Varela, R. Riesen, and P. Roth. Modeling the impact of checkpoints on next-generation systems. In *24th IEEE Conference on Mass Storage Systems and Technologies (MSST 2007)*, pages 30 –46, September 2007.

[33] Y. Pan, G. Dong, and T. Zhang. Exploiting memory device wear-out dynamics to improve NAND flash memory system performance. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies*, February 2011.

[34] K. S. Perumalla. $\mu\pi$: a scalable and transparent system for simulating MPI programs. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools '10, pages 62:1–62:6, ICST, Brussels, Belgium, 2010.

[35] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob. The Structural Simulation Toolkit. *SIGMETRICS Perform. Eval. Rev.*, 38:37–42, March 2011.

[36] R. Rosner, A. Calder, J. Dursi, B. Fryxell, D. Lamb, J. Niemeyer, K. Olson, P. Ricker, F. Timmes, J. Truran, H. Tufo, Y. Young, M. Zingale, E. Lusk, and R. Stevens. Flash code: Studying astrophysical thermonuclear flashes. In *Computing in Science and Engineering*, volume 2, pages 33–41, 2000.

[37] P. Roth. Characterizing the I/O behavior of scientific applications on the Cray XT. In *Proceedings of the 2nd Parallel Data Storage Workshop (PDSW'07)*, November 2007.

[38] B. W. Settlemyer. *A Study of Client-side Caching in Parallel File Systems*. PhD thesis, Clemson University, Clemson, South Carolina, USA, 2009.

[39] J. Shalf. Exascale computing technology challenges. Presented at the HEC FSIO Workshop 2010, August 2010.

[40] 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 the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 42:1–42:12, Piscataway, NJ, USA, 2008. IEEE Press.

[41] G. Shipman, D. Dillow, S. Oral, F. Wang, D. Fuller, J. Hill, and Z. Zhang. Lessons learned in deploying the world's largest scale lustre file system. In *Proceedings of the 52nd Cray User Group Conference (CUG)*, May 2010.

[42] V. Vishwanath, M. Hereld, V. Morozov, and M. Papka. Topology-aware data movement and staging for I/O acceleration on Blue Gene/P supercomputing systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis 2011 (SC11)*, November 2011.

[43] F. Wang, Q. Xin, B. Hong, S. Brandt, E. Miller, D. Long, and T. McLarty. File system workload analysis for large scale scientific computing applications. Technical report, Lawrence Livermore National Laboratory, January 2004.

[44] G. Zheng, G. Gupta, E. Bohm, I. Dooley, and L. V. Kale. Simulating large scale parallel applications using statistical models for sequential execution blocks. In *Proceedings of the 16th International Conference on Parallel and Distributed Systems (ICPADS 2010)*, number 10-15, December 2010.