TO SHARE OR NOT TO SHARE: COMPARING BURST BUFFER ARCHITECTURES

Lei Cao
Bradley W. Settlemyer
High Performance Computing
Los Alamos National Laboratory
Los Alamos, NM, USA
{leicao88124,bws}@lanl.gov

John Bent
Seagate Government Systems
Los Alamos, NM, USA
john.bent@seagategov.com

ABSTRACT

Modern high performance computing platforms employ burst buffers to overcome the I/O bottleneck that limits the scale and efficiency of large-scale parallel computations. Currently there are two competing burst buffer architectures. One is to treat burst buffers as a dedicated shared resource, The other is to integrate burst buffer hardware into each compute node. In this paper we examine the design tradeoffs associated with local and shared, dedicated burst buffer architectures through modeling. By seeding our simulation with realistic workloads, we are able to systematically evaluate the resulting performance of both designs. Our studies validate previous results indicating that storage systems without parity protection can reduce overall time to solution, and further determine that shared burst buffer organizations can result in a 3.5x greater average application I/O throughput compared to local burst buffer configurations.

Keywords: Checkpoint-Restart, Burst Buffers, Storage Systems, File Systems and I/O

1 INTRODUCTION

The overwhelming majority of high-performance computing applications are tightly-coupled, bulk-synchronous parallel simulations that run for days or weeks on supercomputers. Due to both the tight-coupling and the enormous memory footprints used by these application several complexities emerge. One complexity is that the interruption of any process destroys the distributed state of the entire application. Unfortunately, as supercomputers become increasingly powerful, interruptions become more frequent and the size of the distributed state grows. To ensure forward progress these applications use checkpoint-restart, the conceptually simple technique of persisting their distributed state at regular time intervals, such that when an application process fails progress can be restarted from the most recently saved checkpoint. A second complexity is that capturing a consistent view of the distributed state is difficult due to messages in flight. To construct a consistent view of the simulation state, applications quiesce messaging and pause their forward progress for the duration of the checkpoint. These large, frequent bulk-synchronous checkpoints require ever more powerful storage systems capable of ingesting parallel data at massive rates.

Burst buffers, specialized high bandwidth storage tiers composed of relatively expensive solid state media, have arisen to address this performance requirement (Liu, Cope, Carns, Carothers, Ross, Grider, Crume, and Maltzahn 2012, Kimpe, Mohror, Moody, Van Essen, Gokhale, Ross, and de Supinski 2012, Wang, Oral, Wang, Settlemyer, Atchley, and Yu 2014). However, these tiers are too expensive to satisfy the capacity requirements of supercomputers, thus a less expensive, larger disk-based backing storage system with longer data retention and high reliability is also needed. Simple in concept, there are myriad possible configurations for these systems and no systematic means to compare them. For example, while procuring the Department of Energy's first extreme-scale production supercomputer with burst buffers, Los Alamos National Laboratory's (LANL's) Trinity, we encountered several storage system design alternatives but had few capabilities to compare alternatives beyond ensuring vendor offerings satisfied our minimum requirements.

In this paper we describe a small portion of the rich design space of possible burst buffer architectures and compare their efficiency using event-driven simulation. In order to ensure accurate measurements of efficiency, we generate realistic workloads based on the Alliance for Application Performance at Extreme Scale (APEX) Workflows document (LANL, NERSC, and SNL 2016), which describes patterns of I/O exhibited by scientific applications over periods of several months. Our results compare the efficiency of multiple burst buffer designs, and in doing so, allows us to identify several new critical design elements of burst buffer design, including the policies used to manage data staging.

2 RELATED WORK

Due to the observation that multi-tier storage systems are the most cost-effective mechanism by which to provide high bandwidth storage systems to HPC platforms (Bent, Settlemyer, and Grider 2016), the design of burst buffers has been an area of recent interest. Three relevant burst buffer architectures have previously been described, along with their relative strengths and weaknesses (Harms, Oral, Atchley, and Vazhkudai 2016). The compute node-local architecture, with storage devices on each compute node, provides exclusive and interference-free access to the storage devices and linear scaling of bandwidth with the number of compute nodes used by an application. However, common file access techniques, such as many readers or writers sharing a single file (called N-to-1 file access) are difficult to enable. The second model, where burst buffer storage is available on dedicated I/O nodes, supports the N-to-1 access model more easily, and provides opportunities for simpler resilience. However, if more than a single job is running on the supercomputer (a common occurrence), then the jobs must share access to the burst buffer nodes, and risk unpredictable performance due to interference. The final described model co-located the fast storage on the nodes hosting long-term storage. This model creates an off-platform burst buffer, which may not provide the highest absolute performance, but may provide significant ease of use benefits via transparent caching. Although in this paper we only explore the first and second models, our simulator has the flexibility to explore additional paradigms such as network-attached storage, rack-local storage, and hybrids between all of these possibilities.

Both analytical models and simulation have been used to better evaluate the tradeoffs within burst buffer storage systems. An analytical modeling exercise determined that unlike traditional HPC storage systems, a burst buffer provides the most forward progress by providing no reliability (Bent, Settlemyer, DeBardeleben, Faibish, Ting, Gupta, and Tzelnic 2015). Although an unreliable burst buffer may repeatedly fail in such a manner that a long running simulation must restart from the very beginning, the probability is quite low and the improved performance provided by disabling parity protection improves overall throughput enough to offset this possibility. In this paper we replicate this result via simulation.

In simulations of a shared burst buffer architecture for the IBM BlueGene/Q supercomputer (Peng, Divanji, Raicu, and Lang 2016), it was determined that network bottlenecks were likely to be the largest performance limiter. In our simulated architecture, it does not appear that network performance is a limiting factor, but the Cray architecture is quite dissimilar from the simulated IBM architecture. Ongoing work using the ROSS simulator (Carothers, Bauer, and Pearce 2000), is similar to ours in that it focuses on creating a set of realistic I/O workloads (Liu, Cope, Carns, Carothers, Ross, Grider, Crume, and Maltzahn 2012). We are not aware of any other work that generates a realistic HPC workload using historical data and detailed user interviews such as those summarized in the APEX workflow documentation.

Table 1: The subset of the workload information provided by the APEX workflows document used to construct our simulator workload.

Workflow	EAP	LAP	Silverton	VPIC
Workload Percentage	60	5	15	10
Walltime (hours)	984	20	192	144
Hero Run Cores	65536	32768	131072	65536
Routine Number of Cores	32768	4096	32768	32768
Number of Workflow Pipelines per Allocation	125	35	36	24
Anticipated Increase In Problem Size By 2020	10 to 12x	8 to 12x	8 to 16x	8 to 16x
Anticipated Increase In Workflow Pipelines Per Allocation By 2020	2x	1x	1x	1x
Storage APIs	POSIX	POSIX	POSIX	POSIX
Routine Number of Analysis Datasets	100	100	225	150
Checkpoint Style	N to 1	N to 1	N to 1	N to N

3 METHODS

The goal of our simulation is to provide high quality estimates of the performance trade offs associated with several candidate burst buffer architectures. While in our past work we used analytical methods to evaluate storage system design points (such as reliability), here we use simulation to better capture the performance dynamics in the presence of randomly generated errors and faults.

We implemented our simulator based on the Simpy discrete-event simulation framework (Matloff 2008). Events operate on the following entities: a scheduler, jobs, compute nodes, burst buffers nodes, a fault generator, and a parallel file system. Note that burst buffer nodes are only included when simulating shared burst buffer configurations; for local configurations, the burst buffers are on the compute nodes and thus not represented independently by the simulator.

3.1 Workload Generation

In order to build a workload generator for our simulation, we have mined requirements from the APEX Workflows white paper (LANL, NERSC, and SNL 2016). One limitation of the APEX Workflow analysis is that large proportions of the possible laboratory workloads were not well described. While some details for the other sites are provided, and we could speculatively extrapolate projections, we instead chose to simply use the descriptions provided by LANL to construct our entire simulated workload. We recognize that even for LANL's supercomputers this is an overly simplistic assumption as fully two-thirds of LANL's computational cycles are generated by external users.

Table 1 shows a subset of the LANL workload information provided in the workflow document. The APEX workflows paper goes into great detail describing the concepts of campaigns, workflows, pipelines, jobs, and checkpoint strategies. In this paper we will briefly define a *workflow* as the computational process a scientist uses to answer some scientific inquiry. Within a scientific workflow, the scientist then uses some number of workflow *pipelines* (Thain, Bent, Arpaci-Dusseau, Arpaci-Dusseau, and Livny 2003), which are dependent sets of jobs, to simulate and evaluate their hypothesis. For example, a cosmologist may be interested in exploring relativistic particle trajectories as two plasmas move through space. The workflow then would use the Vector Particle-in-Cell (VPIC) code to simulate the plasmas and particles, and the workflow may require multiple pipelines as the cosmologist explores multiple types of plasma intersections and collisions. Each of the initial plasma setup pipelines may take weeks of computational time, though multiple pipelines can be executed in parallel (as large series of parallel jobs).

The LANL workload includes 4 major workflows (EAP, LAP, Silverton, and VPIC). Each of these codes can be used to explore multiple physical phenomena; however, the APEX descriptions indicate that the scale of the computational jobs is consistent per pipeline, and the I/O requirements are approximately fixed for each pipeline. In order to generate computational jobs approximating the provided workload, we have constructed a random workload generator that creates 60 workflows that preserve the workflow percentages and the job size distribution described by the APEX workflows. We do not vary the checkpoint sizes, though we are aware that some codes simulate varying levels of entropy over time.

Finally, we have also simplified our workload evaluation by only simulating checkpoint/restart dumps. Note that the APEX authors also described some of the additional outputs for analysis data and emphasize that analysis data is scientifically valuable in and of itself. Therefore a burst buffer may also benefit the input/output of analysis data; however, even without a fast storage subsystem, analysis data will be generated and stored. Further, the workflows described for analysis data are much more complicated than checkpoint/restart interactions, and thus its not clear that our efforts could adequately characterize interactions for scientific analysis data. Because the authors were members of the APEX Workflows team, our future work will include improving the analysis workflows to further improve our simulation results.

3.1.1 Basic Simulator Execution

The simulation begins by selecting a job from the queue of available jobs. As described above, each job includes requirements for the number of processors, the requested runtime, and the percentage of memory required to create a checkpoint. Job's are scheduled onto the simulated cluster using a first-fit algorithm. While simple, first-fit achieves a high degree of system utilization and is appropriate when we are not evaluating scheduling efficiency metrics such as job turnaround time.

The jobs are then simulated as a series of compute cycles followed by checkpoint phases that store data into the burst buffer system. Checkpoints are created at the near-optimal interval as described by Young and Daly (Daly 2006). As enforced by policy on Trinity, no job is allowed to request a "walltime" longer than 24 hours, so pipelines that require weeks of system time must be decomposed into shorter running jobs. While the first job of a campaign pipeline does not restore itself from an existing checkpoint, all subsequent jobs in that pipeline must read the most recent checkpoint file and restart from that point in the simulation.

We also provide a Poisson-based fault generation process that interrupts system nodes (both compute and burst buffer) causing any process currently using that node to fail. In the case of a failed burst buffer node, the checkpoint process fails; however, the application process continues and will attempt to write an additional checkpoint in the future. When a compute node fails, the entire application must stop, and a job is immediately inserted at the head of the queue to attempt to recoup the progress from the last checkpoint. We used the empirically observed mean time to interrupt (MTTI) from the Trinity supercomputer to parameterize the random arrival process with each node having an equivalent probability of failure.

3.2 Shared Burst Buffer Simulation

For shared burst buffers, simulation of flows both in and out of the burst buffer are critical. Each burst buffer polls for incoming requests. For each request received, it starts an *executor* process to handle that request. The processing time of a request is computed using the minimum of the burst buffer bandwidth and the compute node bandwidth. A burst buffer node's bandwidth is affected by the number of its concurrent requests; a compute node's bandwidth is affected by the number of its burst buffer partitions. A burst buffer node will notify all of its active executors that there is a change in its bandwidth when either a new executor begins or an existing executor finishes. As compute nodes write to multiple burst buffer nodes they split their available bandwidth evenly across them. When any write finishes, the available bandwidth is re-partitioned across the remaining writes and those burst buffers are informed that additional client bandwidth is available which may cause those writes to speed up if those burst buffers have available excess bandwidth.

3.3 Local Burst Buffer Simulation Flow

The local burst buffer simulation is very similar to the shared burst buffer simulation, but with a much simpler implementation. There are no separate burst buffer nodes; instead each compute node has an SSD whose bandwidth comes from the input parameter. When there is a fault, this fault will bring down both the compute node and the burst buffer on that node.

3.4 Simulation Configuration

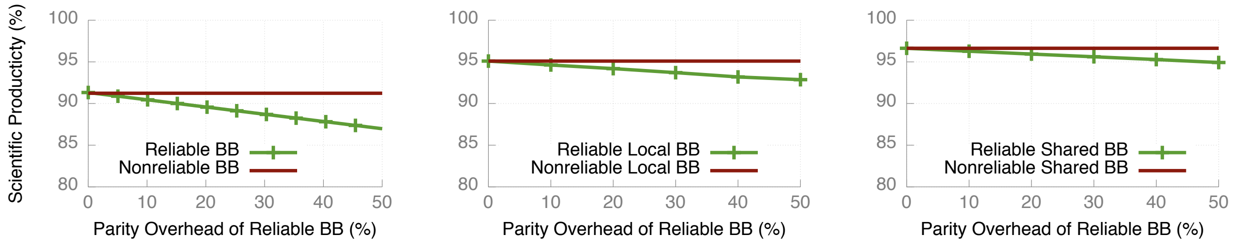
Table 2: Simulation parameters based on LANL's Trinity Supercomputer. Parameters which were varied in order to study their effects are in red italics.

Number of Compute Nodes	9500
Compute Node Memory	128 GB
Compute Node Network Bandwidth	15 GBs
Burst Buffer Type	Local | Shared
Number of Burst Buffer Nodes	If local 9500; if shared 276
Aggregate Burst Buffer Read Bandwidth	1600 GBs
Aggregate Burst Buffer Write Bandwidth	1400 GBs
Parity Overhead	0 | 10 | 20 | 30 | 40 | 50
Burst Buffer Stripes per File	If local 1; if shared 2
Fault Rate	One node per day
Node Recovery Time	1 second | 1 hour

The specification for all simulated shared and local burst buffer architectures is listed in Table 2. Aggregate burst buffer performance is based on empirical measurements using IOR (Various 2016). Similarly, the fault rate is based on the measurement of Trinity's mean time to interrupt (23.8 hours). In order to compare local and shared burst buffer architectures fairly, our local burst buffer simulation splits the aggregate burst buffer bandwidth evenly between each compute node.

4 RESULTS

As discussed in Section 1, our goal in building a burst buffer simulator is to explore various hardware and software design alternatives. Within this study we are limiting ourselves to an exploration of just two possible burst buffer architectures: node-local and shared I/O nodes; however, even within that constraint we have identified several interesting areas of evaluation. In particular we are interested in examining the system impacts related to three different factors. We first examine the impact of whether data in burst buffers is protected with parity, then examine the impact of how long failed nodes take to recover, and finish our evaluation by examining the impact of whether burst buffers are shared or local. The parameters studied are those shown in italicized red in Table 2.



(a) Analytical Model (b) Simulated Local BB (c) Simulated Shared BB

Figure 1: These 3 graphs show the percentage of time dedicated to checkpoint/restart for different burst buffer reliability overheads. Graph (a) shows the analytical results derived in an earlier work, while (b) and (c) show the checkpoint overheads for local and shared burst buffers, respectively. We can see that the simulation results support the earlier analytical result in finding that unreliable burst buffers reduce the overhead related to checkpoint/restart.

4.1 To Protect or Not To Protect

In this portion of our study, we explore the effects of using parity to protect data within the burst buffers. We use BBSim to revisit our earlier analytical study (see Section 2) which showed that unreliable burst buffers (i.e. those that do not add parity protection to data) outperform reliable burst buffers because the performance overhead of adding parity protection outweighs the reliability benefits.

To measure performance, we use the metric of *scientific productivity* which is the amount of time that a job makes forward progress divided by the total amount of time that the job is running. When measuring time spent making forward progress, we exclude checkpointing, restarting, and work that was redone due to failure recovery. Time spent checkpointing and restarting is work required only because the platform is unreliable. The application developers have no interest in checkpoint/restart; rather it is required because long-running simulations cannot complete in the time between job interruptions. Because checkpoint/restart is such a popular technique, some scientists have resorted to analyzing checkpoint files as a sort of crude analysis data, but our current simulation studies are ignoring the costs related to analysis data.

Similarly, we must consider time spent reading the checkpoint data in order to restore the application state as pure overhead. This is slightly unfair as LANL's HPC administrators enforce a scheduler policy that limits jobs to 24 hours of wall clock time for a variety of reasons (fair-sharing, routine maintenance, etc.), and thus restoring from checkpoints is mandatory for any scientific simulation that runs for longer than 24 hours. However, from a pure scientific productivity standpoint we cannot consider time spent reading the most recent checkpoint as directly advancing scientific progress.

Finally, we exclude re-work. Re-work occurs when a job experiences a fault and must be restarted from a prior checkpoint. Although re-work is valid scientific simulation work, it is overhead required only due to the unreliability of the platform and the discrete nature of restoring progress via checkpoint/restart.

In our earlier analytical work, we noted that given a fixed quantity of storage resources, the system could be designed to dedicate some of the resources to storing parity data to improve the system reliability at the cost of some storage system performance (e.g. a 10% parity overhead would result in a 10% storage system slowdown). Although an unreliable storage system will lead to lost checkpoints that result in larger quantities of re-work, the models found that checkpoint overhead was lowest with an unreliable burst buffer due to the much higher incidence of checkpointing to re-work. In Figure 1(a) we present the results from that analysis along with simulation experiments that examine the reliability overheads for local and shared

Table 3: Overall platform efficiency for burst buffer configurations using a 1 second repair/recovery time and a 1 hour repair/recovery time.

Simulation Configuration	Local, 1 Second	Local, 1 Hour	Shared, 1 Second	Shared, 1 Hour
Total Campaign Node-Hours (In Thousands)	18442.563	18696.147	18831.216	18453.385
Productive Compute Node-Hours (In Thousands)	16216.593	16215.224	16447.527	16448.516
Platform Productivity (%)	87.9	86.7	87.3	89.1

burst buffers (Figures 1(b) and 1(c) respectively). The earlier models assumed shared burst buffers; with our simulator, we can now reinforce that finding and extend it to local burst buffers as well: burst buffers, whether local or shared, should not add parity protection for checkpoint data.

Compared to the analytical results, the simulations find higher scientific productivity due to the smaller percentage of memory stored in each checkpoint for the simulated workloads compared to the assumptions included in the analytical models. Note that unlike the analytical study, our simulation results are unable to explore the effects of a reliable burst buffer with 0% reliability overhead, and thus the lines do not cross.

4.2 Effects of Repair/Recovery Time on Overall Productivity

We explore the repair time of nodes because current heuristics to estimate machine productivity have ignored this parameter (essentially assuming that repairing a failed node takes no time). This metric, called $JMTTI/Delta$, is used when procuring HPC platforms and measures the ratio between the mean time to interrupt for a job running across the entire machine and the time to checkpoint that job. In general, a $JMTTI/Delta$ ratio in excess of 200 results in an estimated platform productivity greater than 90%. That is, even in the face of faults causing job restarts and lost work, a job running across the entire machine should make forward progress 90% of the time when $JMTTI/Delta$ is greater than 200.

Although this simple metric is useful in comparing multiple vendor offerings during a procurement cycle, we have long recognized that it is an incomplete picture of the overall scientific productivity of an HPC platform. In addition to other simplifications, it excludes time lost due to repairing failed nodes. However, typical large HPC platforms can have repair/replace times of up to one hour. Therefore, we wanted to check whether $JMTTI/Delta$ can still achieve 90% platform productivity even with repair times up to an hour.

Table 3 shows the results of our simulation studies using a simple model that allows failed nodes (whether a compute node, burst buffer node, or both) to be repaired following a fault in either 1 second or 1 hour. For local burst buffers, we see that productivity decreases slightly with longer repair times as expected. For shared burst buffers, we see a surprising result that productivity *increases* with longer repair times. Upon closer examination, we discovered that this surprising result was due to randomness within our simulator; specifically, faults within our simulation of 1 hour repair times happened unluckily to disproportionately affect larger jobs causing a larger quantity of total cost work. In other words, this unexpected increase in efficiency is due to noise in the simulator.

We therefore generally conclude that the efficiency differences are not very sensitive to recovery time, and rather the mix of jobs and the mean time to interrupt seem to have dominating impacts on overall platform efficiency. In future workloads, with a majority of jobs using extremely large allocations it is likely that we would need to revisit this analysis, but these simulation results indicate that our current $JMTTI/Delta$ procurement metric is likely sufficient.

Table 4: Job checkpoint bandwidths varying both the burst buffer architecture (shared vs. local) and the storage system reliability (unreliable vs. 20% parity overhead.

Simulation Configuration	Local, Unreliable	Local, 20% Parity	Shared, Unreliable	Shared, 20% Parity
Min Application Checkpoint Bandwidth (GB/s)	18.0	14.8	36.1	28.8
Max Application Checkpoint Bandwidth (GB/s)	1424.8	1139.8	1080.4	884.6
Mean Overall Campaign Checkpoint Bandwidth (GB/s)	206.8	165.6	614.54	485.0
Median Overall Campaign Checkpoint Bandwidth (GB/s)	184.8	147.4	645.3	510.6

4.3 To Share or Not to Share

Finally, we examine the effects of organizing the burst buffer into a set of node-local storage devices or as a collection of shared, dedicated burst buffer nodes. In Table 4 we see that the average checkpoint bandwidth encountered throughout the campaign is approximately 3x greater for a shared burst buffer. This is as expected; jobs running with local burst buffers get the exact same percentage of the burst buffer nodes as they do the compute nodes whereas jobs using a small percentage of the compute nodes can use a larger percentage of the shared burst buffer nodes. A more complicated, and interesting, result is that the maximum bandwidth achieved with local burst buffers is higher than the maximum bandwidth achieved with shared. This is because full system jobs using local burst buffers have no resource contention. Our simulation chooses two random burst buffer nodes for each compute node in the shared burst buffer model; this results in some burst buffer nodes being overloaded. The lesson from this result is that shared burst buffers are better for HPC systems that run many small jobs and that shared burst buffers for systems running large jobs need careful mechanisms to reduce contention.

5 CONCLUSION

In this paper we presented BBSim, a simulator for exploring the various effects on scientific productivity arising from the organization of burst buffer storage system. To generate results that go beyond analytical models we used the APEX workflows document to construct a simulation workload derived from real data center workloads. With BBSim we have determined that for the presented workload repair/recovery time following a fault does not have a significant impact on overall platform efficiency. However, in confirmation of earlier analytically derived results, we determined that unreliable burst buffers result in less checkpoint/restart overhead compared to reliable burst buffer configurations. Further, our simulation results validated this conclusion for both shared burst buffer and local burst buffer configurations. Finally, we determined that shared burst buffers result in better overall application checkpoint bandwidth, providing an average storage system bandwidth 3.5x greater than a similarly configured local burst buffer configuration.

In addition to the results published here we have used BBSim to examine the scientific productivity delivered by future system configuration, including hypothetical configurations not currently fielded. BBSim is one of many steps to move our storage system procurement process toward a quantitative evaluation of scientific productivity. The addition of the simulation of analysis and visualization data set creation and access, as well as more robust simulation of the storage system software are two of the key remaining factors aspects needed to gain greater insight into the effects of storage systems to overall scientific productivity.

ACKNOWLEDGMENTS

This work is funded by the RAMSES project, Office of Advanced Computing Research, U.S. Department of Energy and the Ultrascale Systems Research Center (USRC) at Los Alamos National Laboratory supported by the U.S. Department of Energy contract DE-FC02-06ER25750.

This publication has been assigned the Los Alamos National Laboratory identifier LA-UR-17-20080.

REFERENCES

Bent, J., B. Settlemyer, N. DeBardeleben, S. Faibish, D. Ting, U. Gupta, and P. Tzelnic. 2015, July. "On the Non-Suitability of Non-Volatility". In *7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 15)*. Santa Clara, CA, USENIX Association.

Bent, J., B. Settlemyer, and G. Grider. 2016. "Serving Data to the Lunatic Fringe: The Evolution of HPC Storage". *;login: The USENIX Magazine* vol. 41 (2), pp. 34–39. An optional note.

Carothers, C. D., D. Bauer, and S. Pearce. 2000. "ROSS: A High-performance, Low Memory, Modular Time Warp System". In *Proceedings of the Fourteenth Workshop on Parallel and Distributed Simulation*, PADS '00, pp. 53–60. Washington, DC, USA, IEEE Computer Society.

Daly, J. T. 2006. "A higher order estimate of the optimum checkpoint interval for restart dumps". *Future Gener. Comput. Syst.* vol. 22 (3), pp. 303–312.

Harms, K., H. S. Oral, S. Atchley, and S. S. Vazhkudai. 2016, September. "Impact of Burst Buffer Architectures on Application Portability". Technical report, Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States). Oak Ridge Leadership Computing Facility (OLCF).

Kimpe, D., K. Mohror, A. Moody, B. Van Essen, M. Gokhale, R. Ross, and B. R. de Supinski. 2012. "Integrated In-system Storage Architecture for High Performance Computing". In *Proceedings of the 2Nd International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '12, pp. 4:1–4:6. New York, NY, USA, ACM.

LANL, NERSC, and SNL. 2016, March. "APEX Workflows". Technical report, Los Alamos National Laboratory (LANL), National Energy Research Scientific Computing Center (NERSC), Sandia National Laboratory (SNL).

Liu, N., J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn. 2012. "On the role of burst buffers in leadership-class storage systems". In *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–11. IEEE.

Matloff, N. 2008. "Introduction to discrete-event simulation and the simpy language". *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August* vol. 2, pp. 2009.

Peng, J., S. Divanji, I. Raicu, and M. Lang. 2016. "Simulating the Burst Buffer Storage Architecture on an IBM BlueGene/Q Supercomputer".

Thain, D., J. Bent, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. Livny. 2003, June. "Pipeline and Batch Sharing in Grid Workloads". In *Proceedings of High-Performance Distributed Computing (HPDC-12)*. Seattle, Washington.

Various 2016. "IOR HPC Benchmark".

Wang, T., S. Oral, Y. Wang, B. Settlemyer, S. Atchley, and W. Yu. 2014, Oct. "BurstMem: A high-performance burst buffer system for scientific applications". In *Big Data (Big Data), 2014 IEEE International Conference on*, pp. 71–79.

AUTHOR BIOGRAPHIES

LEI CAO is a post master student at New Mexico Consortium. He holds a M.S. degree in Electrical and Computer Engineering from Carnegie Mellon University. His research interest is in HPC storage systems. His email address is leicao88124@lanl.gov

BRADLEY W. SETTLEMYER is a storage systems research and systems programmer specializing in high performance computing. He received his Ph.D in computer engineering from Clemson University in 2009 and works as a research scientist in Los Alamos National Laboratory's HPC Design group. He has published papers on emerging storage systems, long distance data movement, network modeling, and storage systm algorithms. His email address is bws@lanl.gov

JOHN BENT is the Chief Architect at Seagate Government Solutions and has researched storage and IO throughout his career. His recent focus is on parallel storage systems for High Performance Computing. He holds a Ph.D in computer science from the University of Wisconsin-Madison.