# On the Use of Burst Buffers for Accelerating Data-Intensive Scientific Workflows

Rafael Ferreira da Silva
University of Southern California
Information Sciences Institute
Marina del Rey, CA
rafsilva@isi.edu

Scott Callaghan
University of Southern California
Southern California Earthquake Center
Los Angeles, CA
scottcal@usc.edu

Ewa Deelman
University of Southern California
Information Sciences Institute
Marina del Rey, CA
deelman@isi.edu

## ABSTRACT

Science applications frequently produce and consume large volumes of data, but delivering this data to and from compute resources can be challenging, as parallel file system performance is not keeping up with compute and memory performance. To mitigate this I/O bottleneck, some systems have deployed burst buffers, but their impact on performance for real-world workflow applications is not always clear. In this paper, we examine the impact of burst buffers through the remote-shared, allocatable burst buffers on the Cori system at NERSC. By running a subset of the SCEC CyberShake workflow, a production seismic hazard analysis workflow, we find that using burst buffers offers read and write improvements of about an order of magnitude, and these improvements lead to increased job performance, even for long-running CPU-bound jobs.

## KEYWORDS

Scientific Workflows, Burst Buffers, High-Performance Computing, In Transit Processing

**ACM Reference format:**
Rafael Ferreira da Silva, Scott Callaghan, and Ewa Deelman. 2017. On the Use of Burst Buffers for Accelerating Data-Intensive Scientific Workflows. In *Proceedings of Workflows in Support of Large-Scale Science 2017, Denver, CO, USA, November 2017 (WORKS 2017),* 9 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Today's computational and data science applications process and produce vast amounts of data (from remote sensors, instruments, etc.) for conducting large-scale modeling, simulations, and data analytics. These applications may comprise thousands of computational tasks and process large datasets, which are often distributed and stored on heterogeneous resources. Scientific workflows are a mainstream solution to process complex and large-scale computations involving numerous operations on large datasets efficiently. As a result, they have supported breakthrough research across many domains of science [18, 26].

Typically, scientific workflows are described as a directed-acyclic graph (DAG), whose nodes represent workflow tasks that are linked via dataflow edges, thus prescribing serial or parallel execution of nodes. In this paradigm, a task is executed once all its parent tasks (dependencies) have successfully completed. Although some workflow portions are CPU-intensive, many workflows include post-processing analysis and/or in transit visualization tasks that often process large volumes of data [18]. Traditionally, workflows have used the file system to communicate data between tasks. However, to cope with increasing application demands on I/O operations, solutions targeting *in situ* and *in transit* processing have become mainstream approaches to attenuate I/O performance bottlenecks. While in situ is well adapted for computations that conform with the data distribution imposed by simulations, in transit processing targets applications where intensive data transfers are required [8].

By augmenting data volumes and processing times, the advent of Big Data and extreme-scale applications have posed novel challenges to the high-performance computing (HPC) community, for both users and solution providers (e.g. workflow management software developers, cyberinfrastructure providers, and hardware manufacturers). In order to meet the computing challenges posed by current and upcoming scientific workflow applications, the next-generation of exascale supercomputers will increase the processing capabilities to over $10^{18}$ Flop/s [31]—memory and disk capacity will also be significantly increased, and new solutions to manage power consumption will be explored. However, the I/O performance of the parallel file system (PFS) is not expected to improve much. For example, the PFS I/O peak performance for the upcoming Summit (Oak Ridge National Laboratory, ORNL) and Aurora (Argonne National Laboratory, ANL) supercomputers will not outperform Titan's (ORNL) performance, despite being six years newer [17].

Burst Buffers (BB) [1, 29, 30] have emerged as a non-volatile storage solution that is positioned between the processors' memory and the PFS, buffering the large volume of data produced by the application at a higher rate than the PFS, while seamlessly and asynchronously draining the data to the PFS. Advantages and limitations of the use of BB for improving I/O performance of single application executions (e.g., a regular job submitted to a batch queue system) have been an active topic of discussion in the past few years [16, 24, 29]. However, there has been little analysis on the use of BB for scientific workflows [6]. In a recent survey study [11], we characterized workflow management systems w.r.t. their ability to handle extreme-scale applications. Although several systems orchestrate the execution of large-scale workflows efficiently, the optimization of I/O throughput is still a steep challenge.

*WORKS 2017, November 2017, Denver, CO, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In this paper, we propose an architectural model for enabling the use of BB for scientific workflows. More specifically, we discuss practical issues and limitations to support an implementation of a BB available on the Cori system at the National Energy Research Scientific Computing Center (NERSC) facility [3]. Using the Pegasus [7] workflow management system, we evaluate the performance gain of a real-world data-intensive workflow (produces/consumes over 550 GB of data) when its intermediate data is stagged in/out to/from the BB. Experimental results show that the use of a burst buffer may significantly improve the average I/O performance for both read and write operations, however parallel efficiency should be carefully considered when deciding whether to manage all the workflow's intermediate data via a BB. In addition, improvements in I/O bandwidth may be limited by the frequency of I/O operations; i.e. draining the data to the PFS may become the bottleneck.

This paper is structured as follows. Section 2 provides background on data-intensive scientific workflows, and an overview of burst buffer architectures. Section 3 presents an overview of the proposed architectural model for using BB for scientific workflow executions. The experimental evaluation with a real world workflow using a large HPC system is presented in Section 4. Section 5 discusses related work, and underlines the contributions of this paper w.r.t. the state-of-the-art. Section 6 concludes the paper, and identifies directions for future research.

# 2 BACKGROUND

## 2.1 Data-Intensive Scientific Workflows

Scientists want to extract the maximum information out of their data—which are often obtained from scientific instruments and processed in large-scale, heterogeneous distributed systems such as campus clusters, clouds, and national cyberinfrastructures such as the Open Science Grid (OSG) and XSEDE. In the era of Big Data Science, applications are producing and consuming ever-growing data sets, and among other demands (e.g., CPU and memory), I/O throughput has become a bottleneck for such applications. For instance, the automated processing of real-time seismic interferometry and earthquake repeater analysis, and the 3D waveform modeling to calculate physics-based probabilistic seismic hazard analysis [9] both have enormous demands of CPU, memory, and I/O—as presented later on in this paper (Section 4.1). That workflow application consumes/produces over 700GB of data. In another example, a bioinformatics workflow for identifying mutational overlaps using data from the 1000 genomes project consumes/produces over 4.4TB of data, and requires over 24TB of memory across all the tasks [10].

In a recent survey on the management of data-intensive workflows [19], several techniques and strategies, including scheduling and parallel processing, are presented on how workflow systems manage data-intensive workflows. Typical techniques include the clustering of workflow tasks to reduce the scheduling overhead, or grouping tasks that use the same set of data thus reducing the number of data movement operations. Data-aware scheduling techniques also target reducing the number of data movement operations and have been proven efficient for high-throughput computing workloads. In the HPC universe, data-aware techniques have also been explored for in situ processing [11, 17]; however, for in transit or post-processing analyses improvement to the I/O throughput is still a requirement.

## 2.2 Burst Buffers

A burst buffer (BB) is a fast, intermediate non-volatile storage layer positioned between the front-end computing processes and the back-end parallel file system. Although the total size of the PFS storage is significantly larger than the storage capability of a burst buffer, the latter has the ability to rapidly absorb the large volume of data generated by the processors, while slowly draining the data to the PFS—the bandwidth into the BB is often much larger than the bandwidth out of it. Conversely, a burst buffer can also be used to stage data from the PFS for data delivery to processors at high speed. The BB concept is not novel; however, it has gained much attention recently due to the increase in complexity and volume of data from modern applications, and cost reductions for flash storage.

Basically, a burst buffer consists of the combination of rapidly accessed persistent memory with its own processing power (e.g., DRAM), and a block of symmetric multi-processor compute units accessible through high-bandwidth links (e.g., PCI Express, or PCIe for short). Although the optimal implementation of burst buffers is still an open question, two main representative architectures have been deployed: the (1) node-local BB, and the (2) remote-shared BB [28]. In a node-local configuration, the BB is co-allocated with the compute nodes, while in a remote-shared configuration, the BB is deployed into I/O nodes with high-connectivity to compute nodes via a high-speed serial connection. Advantages of the local deployment include the ability to linearly scale the BB bandwidth with the number of compute nodes—the drawback of this approach is that write operations to the PFS may negatively impact the application execution due to the required extra computing power to perform the operation. The remote deployment, on the other hand, mitigates this effect since the I/O nodes have their own processing—but this approach may become an impediment under network congestion. Both approaches have already been widely adopted by current HPC facilities, and in forthcoming HPC systems.

*NERSC Burst Buffer.* In this paper, we conduct experiments using computational resources from the National Energy Research Scientific Computing Center (NERSC). NERSC's burst buffer has been deployed on Cori, a petascale HPC system and #6 on the June 2017 Top500 list [1]. The NERSC BB is based on Cray DataWarp [15], Cray's implementation of the BB concept (Figure 1). A DataWarp node is a Cray XC40 service node directly connected to the Aries network, with PCIe SSD Cards installed on the node. The burst buffer resides on specialized nodes that bridge the internal interconnect of the compute system (Aries HSN) and the storage area network (SAN) fabric of the storage system through the I/O nodes. Each BB node contains a Xeon processor, 64 GB of DDR3 memory, and two 3.2 TB NAND flash SSD modules attached over two PCIe gen3 x8 interfaces, which is attached to a Cray Aries network interconnect over a PCIe gen3 x16 interface. Each node provides approximately 6.4 TB of usable capacity and a peak of approximately 6.5 GB/sec of sequential read and write bandwidth[1].

[1] http://www.nersc.gov/users/computational-systems/cori/burst-buffer/burst-buffer

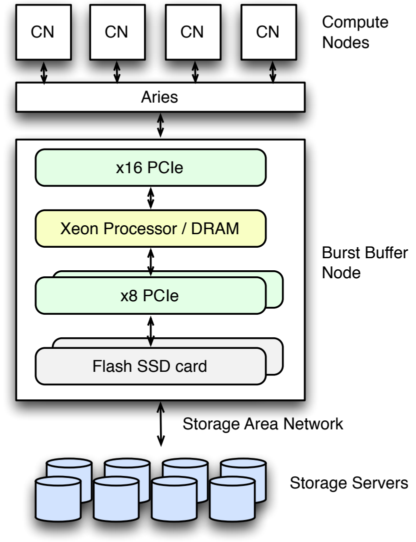

**Figure 1: Architectural overview of a burst-buffer node on Cori at NERSC.**

## 3 MODEL AND DESIGN

Typical workflow executions on HPC systems rely on the underlying parallel file system for staging the computational data to the compute nodes where the workflow tasks are running. As previously discussed, the increasing complexity of current and forthcoming workflow applications, in particular the production and consumption of large volumes of data, imposes challenges for current and upcoming systems, since the performance of the PFS has not increased to the same extent as computing and memory capabilities. As a result, technologies such as burst buffers have emerged as a solution to mitigate this effect, in particular for in transit and post-processing applications.

A current trend in HPC applications is the shifting of the application paradigm towards in memory approaches (e.g., in situ processing). In spite of impressive achievements to date, non-intrusive approaches are still not available. More specifically, numerous workflow applications are composed of legacy codes [26], and thus changing the code to fit modern paradigms is improbable (in some cases, source codes for well established legacy applications are not even available any longer). Therefore, workflow management systems should provide mechanisms to improve the execution of such applications by leveraging state-of-the-art built-in system solutions. In this paper, we propose a practical approach for enabling burst buffer usage for scientific workflows via a non-intrusive method. Our model seeks to abstract configuration and parameter specificities for using burst buffers. In order to enable such seamless use of BB within workflow applications, we argue that a workflow system should automate the following steps:

(1) Burst buffer reservations (either persistent or scratch) should be automatically handled by the workflow management system. This operation includes reservation creation and release, as well as stage in and stage out operations for transient reservations. For such types of reservations, the workflow system needs to implement stage in/out operations at the beginning/end of each job execution.
(2) Workflow systems should automatically map the workflow execution directory (typically known as the execution scratch directory) to the burst buffer reservation. Hence, no changes to the application code are necessary, and the application job directly writes its output to the burst buffer reservation.
(3) I/O read operations should be performed directly from the burst buffer. To this end, the workflow system should make read and write operations from the BB transparent to the application. A simple approach to achieve such transparency is to point the execution directory to the BB reservation (see item above), or to automatically create symbolic links to data endpoints into the burst buffer.

In this paper, we opt for using persistent reservations since stage in/out operations do not need to be performed for intermediate files (reducing the number of data movement operations between the PFS and the BB reservation, and vice-versa), which also facilitates its deployment and management. It is noteworthy that persistent reservations mitigate the burden for coordinating stage in/out data to/from the BB reservation, which may also impact the job execution. On the other hand, for HPC systems where queueing times are systematically long, the cost of stage in/out operation may be negligible, and provisioning BB as late as job start time may yield better overall BB utilization at the system level.

## 4 EXPERIMENTAL EVALUATION

### 4.1 Target Scientific Workflow Application

As part of its research program of earthquake system science, the Southern California Earthquake Center (SCEC) [25] has developed CyberShake [14], a high performance computing software platform that uses 3D waveform modeling to compute physics-based probabilistic seismic hazard analysis (PSHA) estimates for California. CyberShake performs PSHA by first generating a velocity mesh populated with material properties, then using this mesh as input to an anelastic wave propagation code, AWP-ODC-SGT, which generates Strain Green Tensors (SGTs). This is followed by post-processing, in which the SGTs are convolved with slip time histories for each of about 500,000 different earthquakes to generate synthetic seismograms for each event. The seismograms are further processed to obtain intensity measures, such as peak spectral acceleration, which are combined with the probability of each earthquake, obtained from the UCERF2 earthquake rupture forecast[23], to obtain a hazard curve relating ground motion intensities to probability of exceedance. Hazard curves from many (200–400) geographically dispersed locations can be interpolated to produce a hazard map, communicating regional hazard (Figure 2).

For the purposes of exploring burst buffer performance and impact, we focused primarily on the two CyberShake job types which together account for 97% of the compute time: the wave propagation code AWP-ODC-SGT, and the post-processing code DirectSynth which synthesizes seismograms and produces intensity measures. Although the DirectSynth post-processing could theoretically be performed *in situ*, in practice these two codes are often run on different computational systems depending on node type. They were

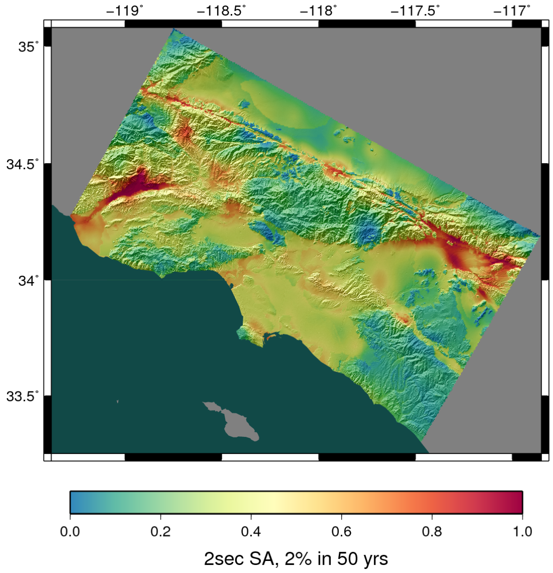

**Figure 2: CyberShake hazard map for Southern California, showing the spectral accelerations at a 2-second period exceeded with a probability of 2% in 50 years.**

also written and are maintained by different developers, making it undesirable to combine the two jobs to enable *in situ* processing.

*AWP-ODC-SGT.* The AWP-ODC-SGT code is a modified version of AWP-ODC, an anelastic wave propagation MPI CPU code developed within the SCEC community and has demonstrated excellent scalability at large core counts (over 10,000 cores) [5]. It takes as input a velocity mesh of about 10 billion points, as well as some small parameter files. For this experiment, we selected a representative simulation, which requires about an hour on 313 Cori nodes, and produces ∼ 275 GB of output. Two of these simulations, one for each horizontal component, must be run in order to produce the pair of SGTs needed for CyberShake post-processing (i.e., `fx.sgt` and `fy.sgt`, a total of ∼ 550 GB) for a single geographic site.

*DirectSynth.* The DirectSynth code is an MPI code, which performs seismic reciprocity calculations. It takes as input a list of fault ruptures and the SGTs generated by AWP-ODC-SGT. From each rupture 10–600 individual earthquakes, which vary in slip and hypocenter location are created, and the slip time history for each earthquake is convolved with the SGTs to produce a two-component seismogram. DirectSynth code follows the master-worker paradigm, in which a task manager reads in the list of ruptures, creates a queue of seismogram synthesis tasks, and then communicates the tasks to the workers via MPI. Processes within the DirectSynth job, the SGT handlers, each read in part of the SGT files, accounting for the majority of data read. Worker processes request and receive the SGTs needed for the convolution from the SGT handlers over MPI. Output data is forwarded to an aggregator, which in total writes 4 files per rupture totaling about 4 MB. For this paper, we selected a CyberShake site with about 5,700 ruptures, resulting in about 23,000 files totaling about 22 GB. Running on 64 Cori nodes, this job takes about 8 hours to complete and produces the outputs CyberShake requires for a single geographic site.

## 4.2 Workflow Implementation

Pegasus-WMS [7] provides the necessary abstractions for scientists to create workflows and allows for transparent execution of these workflows on a range of compute platforms including campus clusters, clouds, and across national cyberinfrastructures. Since its inception, Pegasus has become an integral part of the production scientific computing landscape in several scientific communities. During execution, Pegasus translates an abstract resource-independent workflow into an executable workflow, determining the specific executables, data, and computational resources required for the execution. Workflow execution with Pegasus includes data management, monitoring, and failure handling, and is managed by HTCondor DAGMan [13]. Individual workflow tasks are managed by a task scheduler (HTCondor [27]), which supervises task execution on local and remote resources.

SCEC has used Pegasus to create, plan, and run CyberShake workflows for over a decade. Since the complete end-to-end execution of the workflow requires tens of thousands of CPU hours, we have implemented a smaller version which includes the two CyberShake jobs we are using in our test[2]. Figure 3 shows a graphical representation of the workflow jobs with data and control dependencies. The workflow is composed of two tightly-coupled parallel jobs (`SGT_generator`, i.e. AWP-ODC-SGT; and `direct_synth`), and two *system* jobs (`bb_setup` and `bb_delete`). The computational jobs operate as described in the previous section. For runs utilizing the BB, the `SGT_generator` job writes to the BB (instead of directly to the disk), while the `direct_synth` job reads from it. The system jobs are standalone jobs used to perform management operations in the burst buffer—for this experiment the first job creates a persistent reservation, and the second releases it.

At NERSC, in order to create a BB reservation, one needs to submit a regular standalone job to the batch system, which includes the set of directives to spawn a new BB reservation (either as scratch or persistent), e.g., `#BB create_persistent name=myreservation`. Although the burst buffer reservation creation process is performed upon job scheduling[3], the job remains in the queue until its execution (as any regular batch job). In our workflow model (Figure 3), the `SGT_generator` job would only start to run once the `bb_setup` job is completed—even though the BB reservation may have been already up and running for many hours. Not only may this negatively impact the workflow makespan, it may also result in idle cycles for the BB. To circumvent this issue, we have leveraged DAGMan's `PRE script` concept, which allows jobs to specify processing that will be done before the job submission. We removed the control dependency between BB creation and the first computing job, and defined a PRE script to the `SGT_generator` job that checks the state of the BB reservation creation using the `scontrol` command. Once the reservation is up and running, DAGMan proceeds with the job submission to HTCondor. In this approach, the control dependency

[2] Available online at https://github.com/rafaelfsilva/bb-workflow

[3] As soon as the scheduler reads the job, the Burst Buffer resource is scheduled, even though the job has not yet executed (http://www.nersc.gov/users/computational-systems/cori/burst-buffer/example-batch-scripts/).

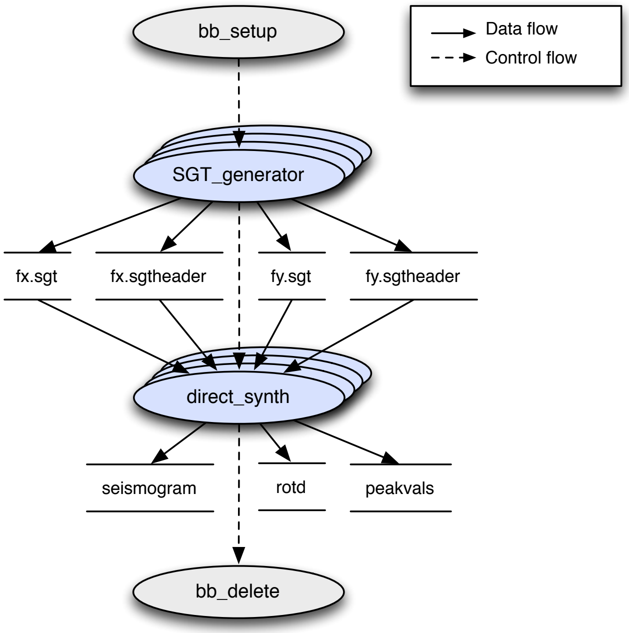

**Figure 3: A general representation of the CyberShake test workflow.**

is represented by the verification step of the PRE script, which triggers the job submission. As a result, the SGT_generator job is submitted as soon as the BB reservation is enabled.

### 4.3 Experiment Conditions

Experiments are conducted with Cori, a Cray XC40 system at NERSC. Cori consists of two partitions, one with Intel Xeon Haswell processors (Phase I, peak performance of 2.3 PFlops) and another with Intel Xeon Phi Knights Landing (KNL) processors (Phase II, peak performance of 29.1 PFlops). For this work, we used the Haswell partition, where each node is composed of 32 cores per node on two 16-core Haswell processors (total of 2,388 cores). Cori also features a 1.8 PB Cray Data Warp Burst Buffer with I/O operating at 1.7 TB/sec. For the experiments conducted in this paper, the bb_setup job creates a persistent BB reservation of 700GB.

Due to our limited allocation of computing cycles at NERSC, and since a single execution of *de facto* SGT_generator (AWP-ODC-SGT) job would consume up to 30% of our current allocation, we developed a synthetic version of the generator job that mimics its I/O behavior for write operations for the SGT files, but significantly reducing the number of CPU cycles needed.

The direct_synth job remains the same. The conclusions of the experimental evaluation discussed in this paper are derived from I/O performance data gathered with Darshan [2]. Darshan is an HPC lightweight I/O profiling tool that captures an accurate picture of I/O behavior (including POSIX IO, MPI-IO, and HDF5 IO) in MPI applications, and is part of the default software stack on Cori.

The goal of this experimental evaluation is to measure the impact of I/O write and read operations to/from the burst buffer for staging in/out intermediate data during the workflow execution. As in our model we create a single BB persistent reservation per workflow run, it is crucial that the I/O throughput yielded by the BB overcomes the application I/O bottleneck. We performed several runs of the CyberShake test workflow (Figure 3) for (1) different numbers of computational nodes (1–313), and (2) different numbers of rupture files (1–5734) to be processed by DirectSynth. The former investigates the ability of the burst buffer to scale with the application's parallel efficiency. The latter studies the impact on the application's makespan when the application becomes more CPU-bound—in our case this is achieved by augmenting the number of rupture files. Although the number of I/O operations also increase in this scenario, the complexity of the computation is significantly augmented as the number of rupture files increase (i.e., the increase on the time spent performing operations in the user space is proportionally larger than the time spent on system operations). For each experiment, we performed several runs of the workflow to obtain measurements within standard errors below 3%.

### 4.4 Results and Discussion

*Overall Write Operations.* Figure 4-*top* shows the average I/O performance estimate for write operations for the synthetic AWP-ODC-SGT (SGT_generator) job for varying numbers of compute nodes on Cori. Note that each node is composed of 32 cores, thus a complete execution (313 nodes) of this job uses 10,016 cores. Performance gain values (Figure 4-*bottom*) represent the average runtime gain for "I/O write" operations (not the task runtime itself) w.r.t. the one-node execution performance. Overall, write operations to the PFS (No-BB) have nearly constant I/O performance; we measured around 900 MiB/s regardless of the number of nodes used. Likely, the PFS automatically balances the I/O bandwidth in order to provide an adequate QoS for all users. Due to slight variations in the measured I/O bandwidth, performance gain values present negligible variations (between 0.95 to 1.0). Workflow runs with the BB, on the other hand, significantly surpass the PFS I/O bandwidth for write operations. Base values obtained for the BB executions (1 node, 32 cores) are over 4,600 MiB/s, and peak values scale up to $\sim 8,200$ MiB/s for 32 nodes (1,024 cores). Increasing the number of nodes ($\geq 64$), we observe a slight drop in the I/O performance due to the large number of concurrent write operations. Although this may be seen as a limitation on the use of burst buffers, the performance degradation is below 10% and the job runtime significantly benefits from the high degree of parallelism.

*Overall Read Operations.* Figure 5-*top* shows the average I/O performance estimate for read operations for the direct_synth job, which consumes the SGT files generated by the SGT_generator job. Typically, CyberShake runs of this job are set to 64 nodes (optimal runtime/parallel efficiency balance). For this experiment, we ran this same job with different numbers of nodes (1 to 128) in order to measure the impact on I/O performance for read operations at different levels of parallelism. Similarly to write operations, read operations from the PFS yield similar performance regardless of the number of nodes used, while the I/O performance varies for reads from the BB—single-node performance of 4,000 MiB/s, peak values up to about 8,000 MiB/s, and then a small dropoff as node counts increase. Although the measured I/O read performance is

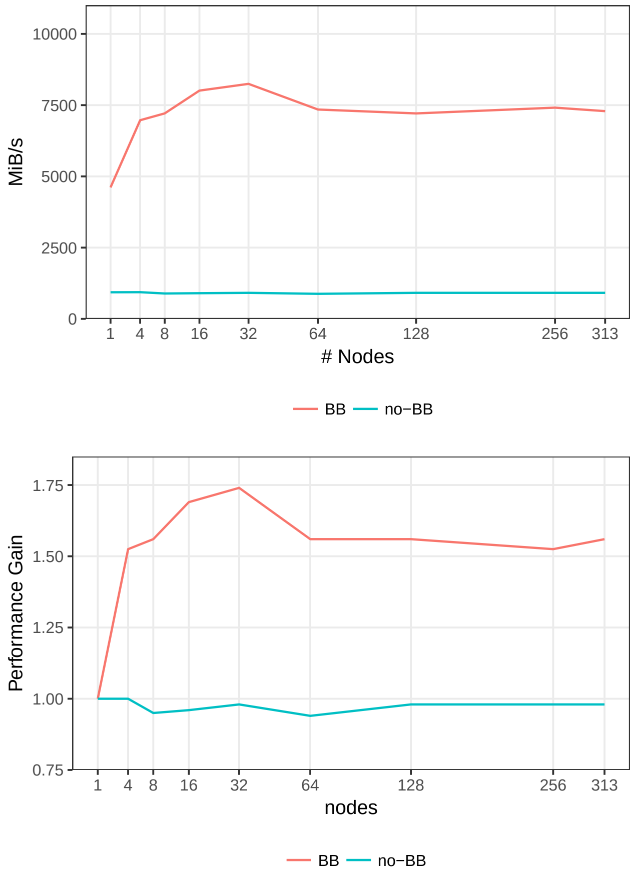

**Figure 4: Average I/O performance estimate for write operations at the MPI-IO layer (top), and average I/O write performance gain (bottom) for the `SGT_generator` job.**

slightly lower than that for write operations (about 5%), we argue that read and write operations achieve similar levels of performance. Notice that I/O write performance gain values (Figure 5-*bottom*) are marginally higher. This result is due to the lower performance yielded by the 1-node execution. Again, we observe a similar small drop in the performance for runs using 64 nodes or above, which may indicate an I/O bottleneck when draining the data to/from the underlying parallel file system. Since queueing time between jobs within a workflow scheduled on Cori may be several hours, a fraction of the files transferred to the BB reservation might be temporarily removed from the BB to improve the efficiency of other users' jobs on the system. Therefore, if the queueing time between two subsequent jobs could be decreased, the observed drop in the performance may be shifted upwards, i.e. I/O contention may occur when using a larger number of nodes.

*I/O Performance per Process.* Figures 6 and 7 show the average time of I/O read operations per process for POSIX and MPI-IO for each horizontal component file (`fx.sgt` and `fy.sgt`), respectively. POSIX operations (Figure 6) represent buffering and synchronization operations with the system. Thus, although there is a visible

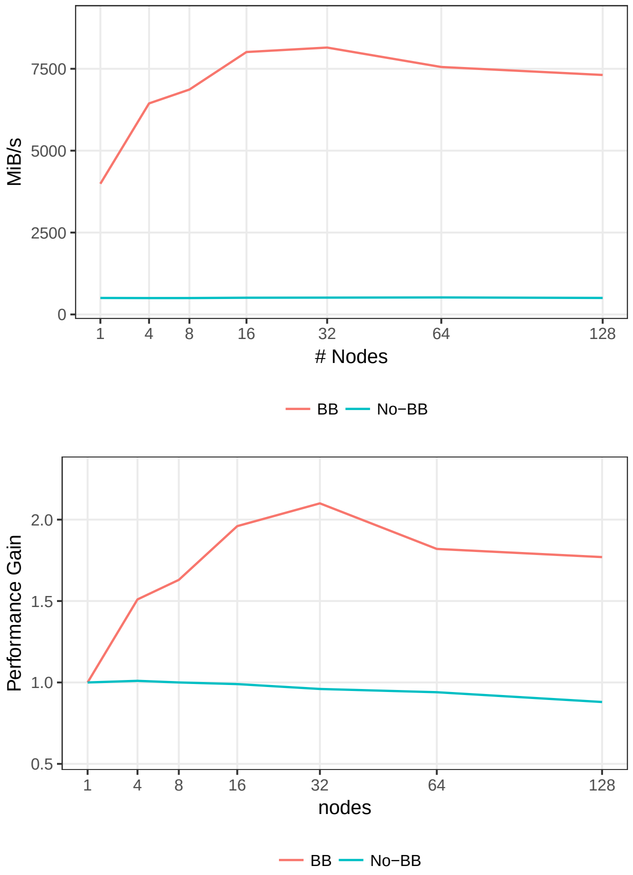

**Figure 5: I/O performance estimate for read operations at the MPI-IO layer (top), and average I/O write performance gain (bottom) for the `direct_synth` job.**

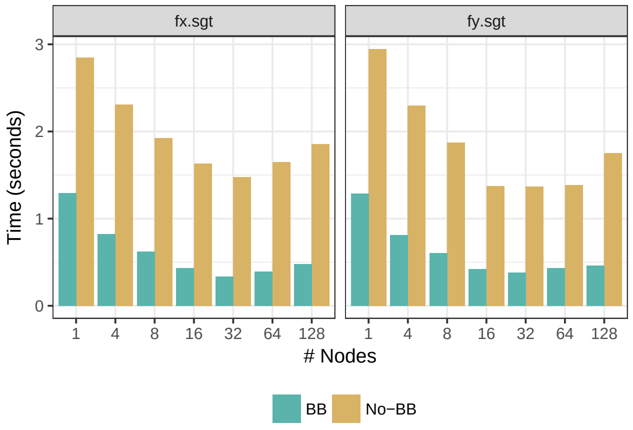

**Figure 6: POSIX module data: Average time consumed in I/O read operations per process for the `direct_synth` job.**

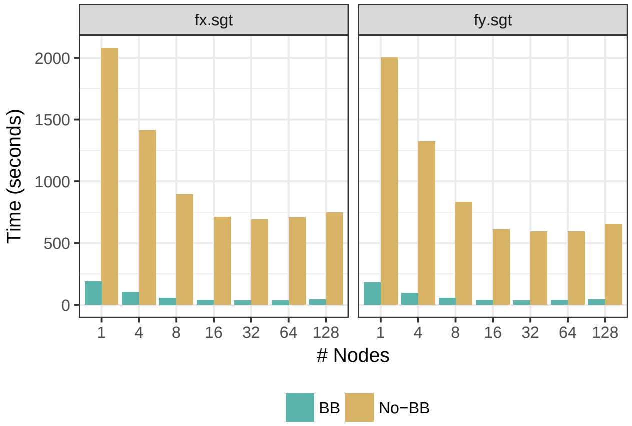

**Figure 7: MPI-IO module data: Average time consumed in I/O read operations per process for the `direct_synth` job.**

difference in the average time consumed in I/O read operations per process between the BB and PFS, these values are negligible when compared to the job's total runtime (approximately 8 hours for 64 nodes). Figure 7 shows the average effective time spent per process performing MPI-IO operations. As expected, the average time consumed in I/O read operations decreases as more process are used. Note that for larger configurations (≥ 32 node), the average time is nearly the same as when running with 16 nodes for the `No-BB` configuration. This behavior is consistent with the I/O performance estimate decline observed in Figure 5. Workflow executions using the BB accelerate I/O read operations up to 10 times in average. It is noteworthy to mention that these averaged values (for up to thousands of cores) may mask slower processes, which may by themselves delay the application execution. In some cases, e.g. 64 nodes, slowest time consumed in I/O read operations can slowdown the application up to 12 times the averaged value. Therefore, we also investigate the distribution of cumulative times of for I/O operations and processing on the user space.

*Cumulative CPU time.* Figure 8 shows the ratio between the time spent in the user (`utime`) and kernel (`stime`) spaces—handling I/O-related interruptions, etc. The use of burst buffers leads the application to a more CPU-intensive pattern. Although executions with 32 nodes yielded the best I/O performance, performance at 64 nodes is similar, suggesting gains in application parallel efficiency would outweigh a slight I/O performance hit at 64 nodes and lead to decreased overall runtime.

*Rupture Files.* As described in Section 4.1, a typical execution of the CyberShake workflow for a selected site in our experiment processes about 5,700 rupture files. Since the number of rupture files may vary for different executions of the workflow, we evaluated the impact on the use of a burst buffer on the application's CPU-boundedness. Figure 9 shows the ratio between the time consumed in the user and kernel spaces for the `direct_synth` job (results for workflow runs with 64 nodes). The processing of rupture files drive most of the CPU (user space) activities for the `direct_synth` job.

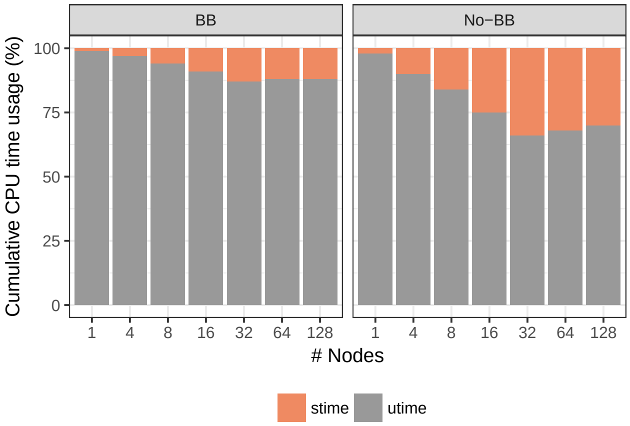

**Figure 8: Ratio between the cumulative time spent in the user (`utime`) and kernel (`stime`) spaces for the `direct_synth` job, for different numbers of nodes.**

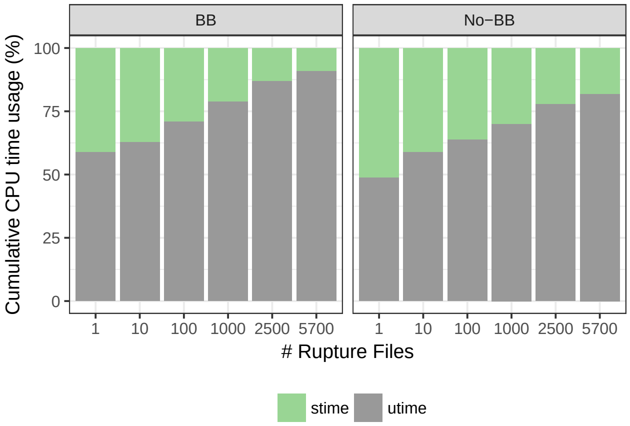

**Figure 9: Ratio between the cumulative time spent in the user (`utime`) and kernel (`stime`) spaces for the `direct_synth` job for different numbers of rupture files.**

Not surprisingly, the more rupture files are used the more CPU-bound the job becomes, yet burst buffers still positively impact the application execution—for our real world workflow, the use of a BB attenuates (about 15%) the I/O processing time of the workflow jobs, for both read and write operations.

## 5 RELATED WORK

Efficient workflow scheduling is an extensively researched topic within the field of scientific workflows. A plethora of studies have targeted, for example, the design and development of cost- and energy-efficient scheduling techniques, while others have focused on the data management aspect of the problem, such as file placement strategies and data-aware scheduling. Numerous survey studies have captured and analyzed the essence of those techniques [4, 19–21]. Although these solutions may improve workflow execution efficiency to some extent, hardware limitations may impose severe barriers, e.g., the workflow execution may be extremely delayed due to I/O contention. An alternative approach is to enable configuration refinement (e.g., change platform conditions). In previous work [22], we investigated scheduling techniques in networked clouds to predict dynamic resource needs using a workflow introspection technique to actuate resource adaptation in response to dynamic workflow needs, which accounts for data flows and network adaptation. However, such approaches cannot be applied to HPC systems.

Burst buffer performance has been thoroughly evaluated in diverse contexts, and its ability to improve I/O throughput for running single parallel applications has been well established [16, 24, 29]. For instance, in [24] an empirical evaluation of a BB implementation with an I/O-bound benchmark application yielded a speedup factor of 20 when compared to the I/O bandwidth from the PFS (in this case GPFS). Their conclusions support the experimental results obtained in this paper, but since they focus on pure I/O-bound applications the impact on the parallel efficiency is neglected. Point solutions at a higher layer of abstraction have also been the target of some studies. BurstMem [30] is a high-performance burst buffer system on top of Memcached [12], which uses a log-structured data organization with indexing for fast I/O absorption and low-latency, semantic-rich data retrieval, coordinated data shuffling for efficient data flushing, and CCI-based communication for high-speed data transfer. BurstFS [29] is a file system solution for node-local burst buffers that provides scalable metadata indexing, co-located I/O delegation, and server-side read clustering and pipelining. Although these systems present solid evaluation and promising results, they are not production-ready. Additionally, NERSC's BB follows a remote-shared pattern, making it incompatible with the use of BurstFS.

In the area of workflow scheduling, Herbein et al. [16] proposed an I/O-aware scheduling technique that consumes a model of links between all levels in the storage hierarchy, and uses this model at schedule time to avoid I/O contention. Experimental results show that their technique mitigates all I/O contention on the system, regardless of the level of underprovisioning. Unfortunately, the approach evaluation is limited to an emulated environment for an FCFS scheduler with support for EASY backfill, which is not production-ready and could not be used to run our real-world data-intensive application. The pioneer work on workflow performance characterization using burst buffers was presented in [6], where two workflow applications from LBNL running on Cori are evaluated. This work was a first step towards the efficient use of BB for scientific workflows. Our contributions in this paper advances this previous work in the following ways: (1) we evaluate a very large data-intensive real-world workflow (consumes/generates over 550 GB of data); (2) we compare the performance gain for using a BB; and (3) we measure the impact of BB at different levels of application parallelism.

## 6 CONCLUSIONS

In this paper, we explored the impact of burst buffers on performance of a real-world scientific workflow application, SCEC CyberShake. Using a software stack including Pegasus-WMS and HTCondor, we ran a workflow on the Cori system at NERSC. The workflow included provisioning and releasing remote-shared BB nodes. We found that for our application, which wrote and read about 550 GB of data, I/O write performance was improved by a factor of 9, and I/O read performance by a factor of 15 when burst buffers were used. Performance decreased slightly at node counts above 64, indicating a potential I/O ceiling, which suggests that I/O performance must be balanced with parallel efficiency when using burst buffers with highly parallel applications.

We acknowledge that I/O contention may limit the broad applicability of burst buffers for all workflow applications. However, solutions such as I/O-aware scheduling or in situ processing may also not fulfill all application requirements. Therefore, we intend to investigate the use of combined in situ and in transit analysis [8, 17], as well as consider more intrusive approaches for changing workflow applications and systems to optimize for burst buffer usage. Future work also includes the development of a production solution for workflow systems, in particular Pegasus, to include the functionality outlined in Section 3, abstract the configuration steps for using burst buffers, and simplify burst buffer use for workflow users. We also intend to characterize the CyberShake workflow (and additional applications) on forthcoming HPC systems that will support an optimized version of the node-local pattern.

## ACKNOWLEDGMENTS

This work was funded by DOE contract number #DESC0012636, “Panorama – Predictive Modeling and Diagnostic Monitoring of Extreme Science Workflows”, and by NSF contract number #1664162, “ SI2-SSI: Pegasus: Automating Compute and Data Intensive Science”. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

CyberShake workflow research was supported by the National Science Foundation (NSF) under the OAC SI2-SSI grant #1148493, the OAC SI2-SSI grant #1450451, and EAR grant #1226343. This research was supported by the Southern California Earthquake Center (Contribution No. 7610). SCEC is funded by NSF Cooperative Agreement EAR-1033462 & USGS Cooperative Agreement G12AC20038.

## REFERENCES

[1] Wahid Bhimji, Debbie Bard, Melissa Romanus, David Paul, Andrey Ovsyannikov, Brian Friesen, Matt Bryson, Joaquin Correa, Glenn K Lockwood, Vakho Tsulaia, et al. 2016. Accelerating science with the NERSC burst buffer early user program. *CUG2016 Proceedings* (2016).

[2] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. 2011. Understanding and improving computational science storage access through continuous characterization. *ACM Transactions on Storage (TOS)* 7, 3 (2011), 8.

[3] Cori – NERSC 2017. http://www.nersc.gov/users/computational-systems/cori/. (2017).

[4] Lauro Beltrão Costa, Hao Yang, Emalayan Vairavanathan, Abmar Barros, Ketan Maheshwari, Gilles Fedak, D Katz, Michael Wilde, Matei Ripeanu, and Samer Al-Kiswany. 2015. The case for workflow-aware storage: An opportunity study. *Journal of Grid Computing* 13, 1 (2015), 95–113.

[5] Yifeng Cui, Efecan Poyraz, Jun Zhou, Scott Callaghan, Phil Maechling, Thomas H. Jordan, Liwen Shih, and Po Chen. 2013. Accelerating CyberShake Calculations on XE6/XK7 Platforms of Blue Waters. In *Proceedings of Extreme Scaling Workshop 2013*.

[6] Christopher S Daley, Devarshi Ghoshal, Glenn K Lockwood, Sudip S Dosanjh, Lavanya Ramakrishnan, and Nicholas J Wright. 2016. Performance Characterization of Scientific Workflows for the Optimal Use of Burst Buffers.. In *11th*

*Workflows in Support of Large-Scale Science, WORKS'16*. 69–73.

[7] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, and Kent Wenger. 2015. Pegasus: a Workflow Management System for Science Automation. *Future Generation Computer Systems* 46 (2015), 17–35. https://doi.org/10.1016/j.future.2014.10.008

[8] Matthieu Dreher and Bruno Raffin. 2014. A flexible framework for asynchronous in situ and in transit analytics for scientific simulations. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 277–286.

[9] Rafael Ferreira da Silva, Ewa Deelman, Rosa Filgueira, Karan Vahi, Mats Rynge, Rajiv Mayani, and Benjamin Mayer. 2016. Automating Environmental Computing Applications with Scientific Workflows. In *Environmental Computing Workshop (ECW'16), IEEE 12th International Conference on e-Science*. 400–406. https://doi.org/10.1109/eScience.2016.7870926

[10] Rafael Ferreira da Silva, Rosa Filgueira, Ewa Deelman, Erola Pairo-Castineira, Ian Michael Overton, and Malcolm Atkinson. 2016. Using Simple PID Controllers to Prevent and Mitigate Faults in Scientific Workflows. In *11th Workflows in Support of Large-Scale Science (WORKS'16)*. 15–24.

[11] Rafael Ferreira da Silva, Rosa Filgueira, Ilia Pietri, Ming Jiang, Rizos Sakellariou, and Ewa Deelman. 2017. A Characterization of Workflow Management Systems for Extreme-Scale Applications. *Future Generation Computer Systems* 75 (2017), 228–238. https://doi.org/10.1016/j.future.2017.02.026

[12] Brad Fitzpatrick. 2004. Distributed caching with memcached. *Linux journal* 2004, 124 (2004), 5.

[13] James Frey. 2002. Condor DAGMan: Handling inter-job dependencies. (2002).

[14] Robert Graves, Thomas H Jordan, Scott Callaghan, Ewa Deelman, Edward Field, Gideon Juve, Carl Kesselman, Philip Maechling, Gaurang Mehta, Kevin Milner, et al. 2011. CyberShake: A physics-based seismic hazard model for southern California. *Pure and Applied Geophysics* 168, 3-4 (2011), 367–381.

[15] Dave Henseler, Benjamin Landsteiner, Doug Petesch, Cornell Wright, and Nicholas J Wright. 2016. Architecture and Design of Cray DataWarp. In *Proc. Cray Users' Group Technical Conference (CUG)*.

[16] Stephen Herbein, Dong H Ahn, Don Lipari, Thomas RW Scogland, Marc Stearman, Mark Grondona, Jim Garlick, Becky Springmeyer, and Michela Taufer. 2016. Scalable I/O-Aware Job Scheduling for Burst Buffer Enabled HPC Clusters. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 69–80.

[17] Travis Johnston, Boyu Zhang, Adam Liwo, Silvia Crivelli, and Michela Taufer. 2017. In situ data analytics and indexing of protein trajectories. *Journal of computational chemistry* 38, 16 (2017), 1419–1430.

[18] Chee Sun Liew, Malcolm P Atkinson, Michelle Galea, Tan Fong Ang, Paul Martin, and Jano I Van Hemert. 2016. Scientific workflows: moving across paradigms. *ACM Computing Surveys (CSUR)* 49, 4 (2016), 66.

[19] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. 2015. A survey of data-intensive scientific workflow management. *Journal of Grid Computing* 13, 4 (2015), 457–493.

[20] Li Liu, Miao Zhang, Yuqing Lin, and Liangjuan Qin. 2014. A survey on workflow management and scheduling in cloud computing. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 837–846.

[21] Jianwei Ma, Wanyu Liu, and Tristan Glatard. 2013. A classification of file placement and replication methods on grids. *Future Generation Computer Systems* 29, 6 (2013), 1395–1406.

[22] Anirban Mandal, Paul Ruth, Ilya Baldin, Yufeng Xin, Claris Castillo, Gideon Juve, Mats Rynge, Ewa Deelman, and Jeff Chase. 2015. Adapting Scientific Workflows on Networked Clouds Using Proactive Introspection. In *IEEE/ACM Utility and Cloud Computing (UCC)*. https://doi.org/10.1109/UCC.2015.32

[23] 2007 Working Group on California Earthquake Probabilities. 2008. The Uniform California Earthquake Rupture Forecast, Version 2. (2008). https://pubs.usgs.gov/of/2007/1437/

[24] Wolfram Schenck, Salem El Sayed, Maciej Foszczynski, Wilhelm Homberg, and Dirk Pleiter. 2017. Evaluation and Performance Modeling of a Burst Buffer Solution. *ACM SIGOPS Operating Systems Review* 50, 1 (2017), 12–26.

[25] Southern California Earthquake Center 2017. http://www.scec.org. (2017).

[26] Ian J Taylor, Ewa Deelman, Dennis B Gannon, and Matthew Shields. 2007. *Workflows for e-Science: scientific workflows for grids*. Springer Publishing Company, Incorporated.

[27] Douglas Thain, Todd Tannenbaum, and Miron Livny. 2005. Distributed computing in practice: the Condor experience. *Concurrency and computation: practice and experience* 17, 2-4 (2005), 323–356.

[28] Teng Wang. 2017. *Exploring Novel Burst Buffer Management on Extreme-Scale HPC Systems*. Ph.D. Dissertation. The Florida State University.

[29] Teng Wang, Kathryn Mohror, Adam Moody, Kento Sato, and Weikuan Yu. 2016. An ephemeral burst-buffer file system for scientific applications. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 807–818.

[30] Teng Wang, Sarp Oral, Yandong Wang, Brad Settlemyer, Scott Atchley, and Weikuan Yu. 2014. Burstmem: A high-performance burst buffer system for scientific applications. In *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 71–79.

[31] White House National Strategic Computing Initiative Workshop Proceedings 2015. https://www.nitrd.gov/nsci/files/NSCI2015WorkshopReport06142016.pdf. (2015).