Measuring the Impact of Burst Buffers on Data-Intensive Scientific Workflows

Rafael Ferreira da Silva[a,*], Scott Callaghan[b], Tu Mai Anh Do[a], George Papadimitriou[a], Ewa Deelman[a]

[a] *University of Southern California, Information Sciences Institute, Marina del Rey, CA, USA*
[b] *University of Southern California, Southern California Earthquake Center, Los Angeles, CA, USA*

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 scientific workflow applications is still not 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 two data-intensive workflows, a high-throughput genome analysis workflow, and a subset of the SCEC high-performance CyberShake workflow, a production seismic hazard analysis workflow, we find that using burst buffers offers read and write improvements of an order of magnitude, and these improvements lead to increased job performance, and thereby increased overall workflow performance, even for long-running CPU-bound jobs.

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

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 perform large-scale, complex computations on large datasets efficiently. As a result, they have supported breakthrough research across many domains of science [1, 2].

Typically, scientific workflows are described as a directed-acyclic graph (DAG), whose nodes represent workflow tasks that are linked via dataflow or control 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 I/O-intensive analysis and/or in transit visualization tasks that often process large volumes of data [2]. 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 [3, 4]. 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 [4].

By increasing 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. Users and solution providers (e.g., workflow management software developers, cyberinfrastructure providers, and hardware manufacturers) have to rethink their approach to data management within HPC systems. 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 [5]—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 [6] (Oak Ridge National Laboratory, ORNL) and Aurora [7] (Argonne National Laboratory, ANL) supercomputers does not outperform Titan's (ORNL) performance, despite being six years newer [3].

Burst Buffers (BB) [8, 9, 10] 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 [11, 12, 13, 10, 14]. However, there has been little analysis on the use of BB for scientific workflows [15, 16, 17]. In a recent survey study [18], we characterized workflow management systems with regard to their

*Corresponding address: USC Information Sciences Institute, 4676 Admiralty Way Suite 1001, Marina del Rey, CA, USA, 90292

Email addresses: rafsilva@isi.edu (Rafael Ferreira da Silva), scottcal@usc.edu (Scott Callaghan), tudo@isi.edu (Tu Mai Anh Do), georgpap@isi.edu (George Papadimitriou), deelman@isi.edu (Ewa Deelman)

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 challenge.

In this paper, we propose a generic architectural model for enabling the use of BB for scientific workflows. More specifically, we discuss practical issues and limitations in supporting an implementation of a BB available on the Cori system at the National Energy Research Scientific Computing Center (NERSC) facility [19]. Using the Pegasus [20] workflow management system, we evaluate the performance gain of two real-world data-intensive workflows: (1) a high-throughput multi-threaded workflow that constantly performs I/O read and write operations from/to the BB; and (2) a high-performance workflow that 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 data and 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.

In [21], we have presented an evaluation of NERSC's BB implementation for running a subset of a large-scale HPC workflow. This paper extends the prior work by: (1) providing a more detailed discussion of the architectural model and its initial implementation; (2) conducting a performance comparison (using benchmark) between a BB implementation and a parallel file system; and (3) evaluating the impact of I/O read and write operations from/to the burst buffer for processing I/O bound jobs from a high-throughput genomics workflow.

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 challenges and an overview of an architectural model for using BB for scientific workflow executions. The experimental evaluation of using BB for running scientific I/O-intensive applications with two real world data-intensive workflows using a large HPC system is presented in Section 4. Section 5 discusses related work, and underlines the contributions of this paper with regard to 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) [22] and XSEDE [23]. 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 data, and the 3D waveform modeling for the calculation of physics-based probabilistic seismic hazard analysis [24] have enormous demands of CPU, memory, and I/O—as presented later on in this paper (Section 4.3). 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 [25, 26].

In a recent survey on the management of data-intensive workflows [27], several techniques and strategies, including scheduling and parallel processing are presented about 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 area of HPC, data-aware techniques have also been explored for in situ processing [3, 18]; however, for in transit or post-processing analyses improvement to the I/O throughput is still needed.

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.

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: (1) the node-local BB, and (2) the 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

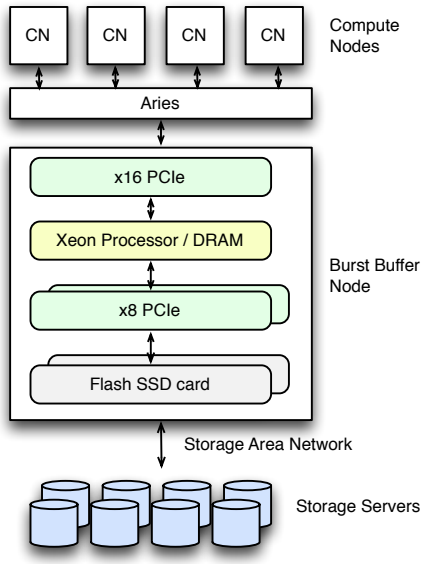

Figure 1: Architectural overview of a burst-buffer node on Cori at NERSC.

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 #12 on the November 2018 Top500 list [9]. The NERSC BB is based on Cray DataWarp [29], 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]. NERSC's burst buffer implementation provides two models for storing data, via a scratch space or a persistent reservation. Scratch spaces are recommended for temporary storage of files during job execution (i.e., files will be removed upon job completion), while in a persistent reservation files that need to be accessed across jobs will be kept in a reserved space after job completion (files are removed only once the reservation is released).

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

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, poses 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 [1], and thus changing the code to fit modern paradigms is improbable (in some cases, source codes for well established legacy applications are no longer available). Therefore, workflow management systems should provide mechanisms to improve the execution of such applications by leveraging state-of-the-art built-in system solutions.

We envision a generic I/O management framework for workflow systems, in which I/O parameterization is presented in an abstracted way to the scientists. In such a model, the workflow management system should automate the deployment and configuration of the underlying system to seamlessly enable in transit or in situ processing (as per user request). For example, a scientist may instrument their code to enable in memory storage processing (e.g., by using tools such as DataSpaces [30]), or use burst buffers for improving the workflow I/O throughput—in particular for scientific workflows where communication between tasks are based on files. A first practical implementation similar to the proposed model was presented in [16], in which the framework coordinates, tracks, and manages the data lifecycle of workflow executions on HPC systems. Here, we expand this model for transparent in transit data management for scientific workflows.

Figure 2 shows an architectural overview of the proposed I/O management framework. Typically, data exchange among workflow jobs/tasks are performed via files that are stored in the local or parallel filesystem. Our proposed model seeks to abstract data exchange handling by automating the data transport layer deployment and configuration process. From the users perspective, in addition to the workflow description and common configuration details (e.g., computing sites, storage resources, etc.), information regarding the data transport layer should also be provided, i.e. whether input, intermediate, and output files are stored in the local disk, parallel filesystem, or a burst buffer. Current workflow systems, e.g. Pegasus, already provide mechanisms to map data from local disks and parallel filesystems by simply providing a path to the storage mapping in the system. Therefore, we argue that support for emerging technologies should follow similar approach.

In order to enable such seamless use of BB within workflow

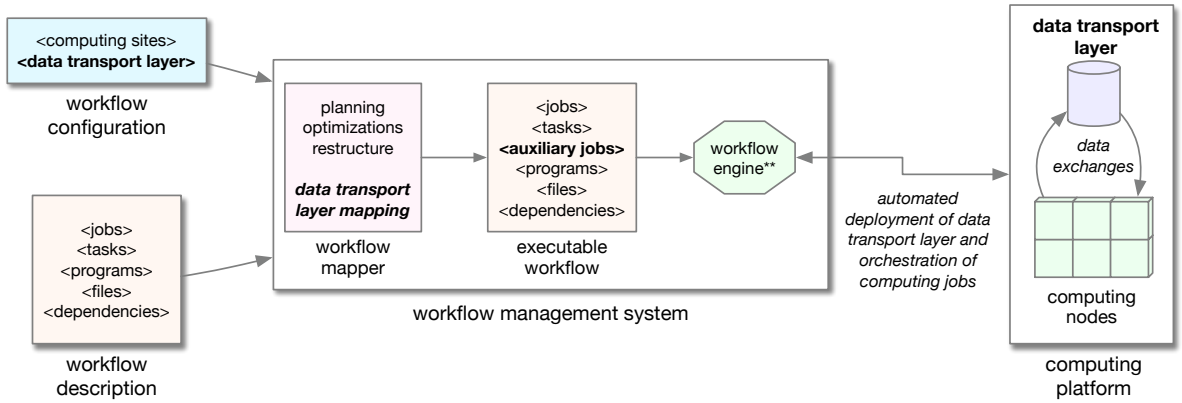

Figure 2: Architectural overview of an I/O management framework for workflow systems. The workflow engine needs to be augmented with mechanisms to automatically deploy and configure the data transport layer according to the user's specification.

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. A more advanced implementation would provide a dynamic solution where persistent and scratch reservations interleave throughout the workflow execution, leading to improved I/O throughput and higher system utlization.

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 (enables support for legacy applications), 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 our model (Figure 2), the data transport layer mapping module of the workflow management system needs to be extended to augment the executable workflow with *auxiliary* jobs for enabling I/O operations to/from the burst buffer. As aforementioned, enabling such support entails extending the executable workflow with additional jobs for setting the burst buffer reservation, performing the necessary adjustments to files and directories paths, and releasing the reservation. Since BB implementation may vary depending on the system and thereby its configuration and usage, it is critical that the workflow mapper component (generates an executable workflow based on an abstract workflow provided by the user or a workflow composition system) abstracts such complexity to insure the fundamental notion of portable workflow descriptions. To this end, the workflow mapper may need implementation-specific software for handling such heterogeneity. In our practical implementation, described below and used for the experimental evaluation presented in Section 4, workflow portability is guaranteed since BB usage is enacted by the workflow system, thus the workflow description only has high-level description of file names.

A key challenge for performing efficient and transparent I/O operations with burst buffers, while boosting its utilization, is to automate the process of creating and releasing reservations (either persistent or temporary). Achieving optimal performance in practice requires sophisticated mechanisms (e.g., online monitoring feedback loops [31], estimate the size of the burst buffer based on application characteristics [32], etc.), which are out of the scope of this paper. Instead, our goal is to demonstrate that even by using simple mechanisms to leverage the capabilities of burst buffers for the execution of in transit workflows, a significant improvement on fine- and coarse-grained workflow metrics can be attained (though BB usage metrics may be unsatisfactory).

3.1. An Initial Practical Implementation

In this paper, we performed a preliminary implementation of the aforementioned model using a popular, well-established workflow management system: Pegasus [20]. Pegasus 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 [33]. Individual work-

flow tasks are managed by a task scheduler (HTCondor [34]), which supervises task execution on local and remote resources.

Our model enables burst buffer usage for scientific workflows via a non-intrusive method, and seeks to abstract configuration and parameter specificities for using burst buffers. 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 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.

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). Although the burst buffer reservation creation process is performed upon job scheduling[2], the job remains in the queue until its execution (as any regular batch job). In an idealized implementation (Figure 3), a compute job would only start to run once the `bb_setup` job is completed. A drawback of such approach is that the BB reservation may have been already up and running for many hours. This may negatively impact the workflow makespan and it may result in idle cycles for the BB. To alleviate this issue, in our implementation with Pegasus 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 dependencies between BB creation and the first computing jobs, and defined a PRE script for each of these first computing jobs 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—note that the `bb_setup` job will still be in the queue when the workflow jobs are released for submission. In this approach, the control dependency (dashed green edge between the `bb_setup` and the first job in Figure 3) is represented by the verification step of the PRE script, which triggers the job submission. As a result, the first set of jobs is submitted as soon as the BB reservation is enabled. An alternative approach to reduce the timespan between reservation creation and actual computing job start would be to submit the computing job to the queue, and upon job start the workflow system would submit the `bb_setup` job (the PRE script would ensure the computing job would not start running until the BB reservation has been enabled).

For the experiments presented in the next section, we have modified the application's configuration files (by adding the BB reservation environment variable to the files paths) to write and read in/output files from/to the burst buffer. A full-fledged reference implementation of the proposed model will be performed in the near future, and will be made available as part of the Pegasus software. Such implementation will provide automated support for the generation of symbolic links based on the presence of input files in the BB reservation, and the mapping of the working directory to the burst buffer.

[2]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).

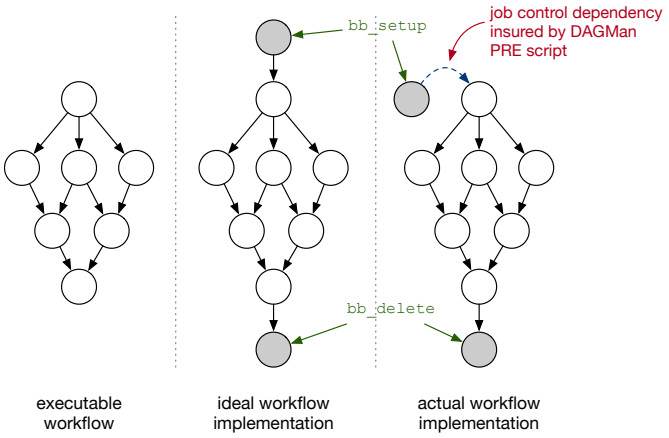

Figure 3: Illustrative example of how burst buffer reservations are created/removed during workflow execution. In an ideal implementation, a setup (resp. delete) job would be added to the beginning (resp. end) of the workflow. In the actual implementation the job control dependency between the BB setup and job execution is performed via a script mechanism to optimize the workflow execution.

4. Experimental Evaluation

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.

Our experimental evaluation is twofold. First, we conduct a simple evaluation benchmark of the NESRC's BB implementation to highlight its performance gain with regard to the parallel file system. Then, we conduct two sets of experiments with real world, large-scale workflows. The first investigates whether NERSC's Cray Data Warp BB implementation may improve the execution performance of a high-throughput data-intensive workflow from the bioinformatics domain; and the second examines whether burst buffers may be used to improve intermediate data handling for a data-intensive high-performance workflow.

4.1. Overall Evaluation of Burst Buffers Performance

Evaluation and performance modeling of Burst Buffers have been conducted extensively by the community, such as in [8, 9, 10, 12, 13]. In this work, we conduct a simple benchmark

evaluation of the NERSC's BB implementation aiming a performance comparison analysis between the burst buffer and the parallel file system—in this case a remote-shared persistent reservation in the burst buffer and the Lustre parallel distributed file system deployed at NERSC.

The conclusions of the experimental evaluation discussed in this paper are derived from I/O performance behavior gathered with Darshan [35]. Darshan is an HPC lightweight I/O profiling tool that captures data for each file opened by the application, including I/O operation counts, common I/O access sizes, cumulative timers, etc. I/O behavior is captured for POSIX IO, MPI-IO, HDF5, and Parallel netCDF data interface layers. Darshan can instrument I/O functions in both statically and dynamically linked executables. Darshan is part of the default software stack on Cori, and is enabled by default.

We have developed a synthetic parallel application that writes/reads blocks of data to/from multiple processes running concurrently. For each block size (i.e., 64, 128, 256, 512, and 1024 MB), we have run the application with 2^n processes (where $n \in [2, 7]$), i.e., a total of 30 experimental scenarios. Note that the execution of a scenario using, for example, a block size of 128MB and 8 processes, will produce/consume 1 GB of data in total. NERSC's Cori system provides nodes with 32 cores (up to 64 processes can run concurrently within a single node via hyper-threading technology), thus runs performed with 128 processes use 2 nodes. For each experimental scenario, we performed several runs of the application to obtain measurements within standard errors below 5%.

4.1.1. Results and Discussion

Figure 4 shows a performance comparison (measured in MiB/s) for *write* operations to the BB (Figure 4-*top*), or the Lustre PFS (Figure 4-*bottom*). Not surprisingly, the BB implementation outperforms the PFS for I/O write operations by a factor of 4. The Lustre PFS yields a nearly constant performance, although a slightly deterioration on the performance is observed for 32 and 64 processes. On the other hand, the BB yields slightly increasing performance as the number of processes and the data block size increase—although the performance is nearly constant for larger block sizes. For 128 processes (i.e., 2 nodes), a considerable spike in the performance is observed. This indicates that for larger block sizes and number of processes there is a saturation on the Aries link between the compute node and the BB node. Similarly to I/O write operations, I/O read operations (Figure 5) performed via BB (Figure 5-*top*) outperforms the PFS (Figure 5-*bottom*) by a factor of 2. In contrast, I/O read operations on the PFS yields similar increasing performance behavior as for the BB, thus a lower performance improvement is observed. It is important to notice that a similar saturation is also observed for I/O read operations on the burst buffer. This preliminary evaluation allows us to ensure the I/O performance of our target large scale (O(100) of processes) workflow applications (presented below) will not be limited by the computational infrastructure.

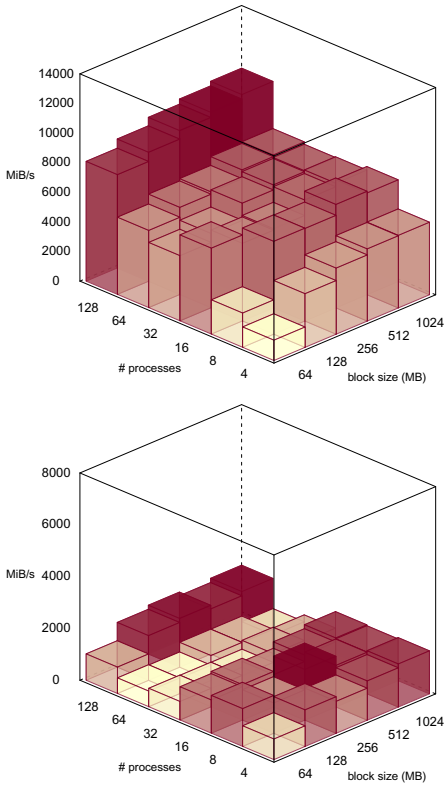

Figure 4: I/O Write: performance comparison of write operations with burst-buffers (top) and the PFS (bottom) at NERSC.

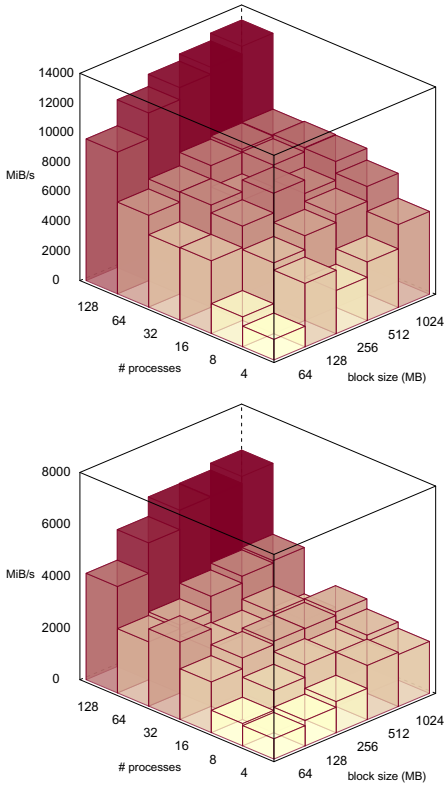

Figure 5: I/O Read: performance comparison of read operations with burst-buffers (top) and the PFS (bottom) at NERSC.

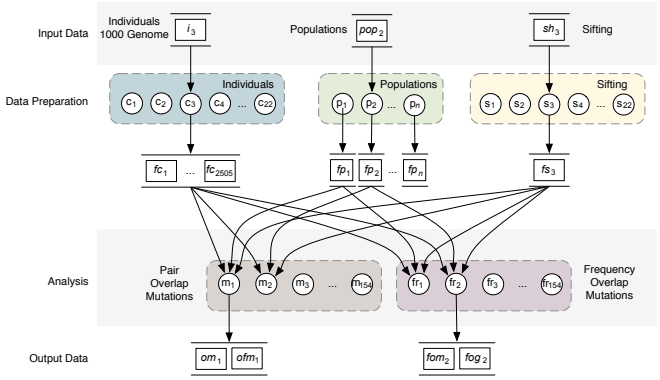

Figure 6: Overview of the 1000 genome sequencing analysis workflow.

4.2. Burst Buffers for High-Throughput Workflows

The goal of this experimental evaluation is to measure the impact of I/O read and write operations from/to the burst buffer for processing I/O bound jobs of a high-throughput data-intensive bioinformatics workflow.

The 1000 genomes project provides a reference for human variation, having reconstructed the genomes of 2,504 individuals across 26 different populations [36]. The test case used in this paper identifies mutational overlaps using data from the 1000 genomes project in order to provide a null distribution for rigorous statistical evaluation of potential disease-related mutations. This test case (Figure 6) has been implemented as a Pegasus workflow [25], and is composed of five different tasks: (1) *individuals* – This task fetches and parses the Phase 3 data from the 1000 genomes project per chromosome; (2) *populations* – The populations task fetches and parses five super populations (African, Mixed American, East Asian, European, and South Asian), and a set of all individuals; (3) *sifting* – This task computes the SIFT scores1 of all of the SNPs (single nucleotide polymorphisms) variants, as computed by the Variant Effect Predictor; (4) *pair overlap mutations* – This task measures the overlap in mutations (SNPs) among pairs of individuals; and (5) *frequency overlap mutations* – This task calculates the frequency of overlapping mutations across subsamples of certain individuals.

The total data footprint for a typical run of the genomics workflow is about 4.4 TB, and requires over 400 GB of RAM (for the largest tasks: `individuals`). A detailed characterization of this workflow is available in [25]. In order to fit an instance of the workflow execution into NERSC's debug queue (see description below), we have pruned the original datasets to process about 10% of the original data (about 11GB per individual dataset, which contains 80,000 records each), and the processing of 2 chromosomes. For this experiment, each workflow is composed of 20 individuals jobs (10 per chromosome), 2 sifting jobs, 14 frequency overlap mutations jobs, and 14 pair overlap mutations jobs.

4.2.1. Experiment Conditions

As described in [25], the turnaround time for computing all jobs from the 1000 genome workflow (i.e., workflow makespan) is dominated by the time to compute the `individuals` jobs—about 98% of the workflow computing processes and I/O operations are performed by these jobs. The individual jobs list all of the Single nucleotide polymorphisms (*SNPs*) variants in the Phase 3 data [36] from the 1000 genome project per chromosome associating them to individuals. An `individuals` job creates output files for each individual of *rs numbers*, where individuals have mutations in at least one of the two alleles. This task is extremely I/O-intensive, where I/O read operations are continuously performed in an individual dataset, while I/O write operations are performed for each identified mutation.

Since the `individuals` jobs are embarrassingly parallel tasks, I/O performance behavior cannot be captured with traditional HPC I/O profiling tools. Therefore, we use the Kickstart [37] profiling tool to gather the job turnaround time and its I/O footprint (i.e., number of bytes read and written). To improve system utilization, for each `individuals` job we split the dataset into smaller chunks of data (10 per job, i.e., 8,000 records per job) to be processed in parallel—we leverage the job clustering capability from the Pegasus [20] workflow management system, a wrapper tool that takes a set of jobs and execute them sequentially or in parallel (via threads) on a single node. For each experiment scenario, we performed several runs of the `individuals` jobs to obtain averaged measurements within standard errors below 2%. The experiments conducted in this section used a persistent BB reservation of 50 GB.

4.2.2. Results and Discussion

Figure 7 shows the average job turnaround time (makespan) and I/O wait for the `individuals` jobs performing I/O read and write operations on the parallel file system (red triangles) and on the burst buffer (black squares). Overall, runs using the BB implementation speed up the `individuals` jobs up to a factor of 2 (Figure 7-*top*). Note that although the `individuals` jobs read from the same two individual dataset, the amount of I/O operations (mostly read operations) may vary per job—thus the variation on the *x*-axis values. Although the datasets are divided into smaller data blocks (each `individuals` job only reads the necessary amount of data for processing), the same data block may be read multiple times—in particular for very large datasets. For instance, an actual execution of a complete version of an `individuals` job of the 1000 genome workflow may process files larger than 300 GB. During execution, an output file is generated for each mutation observed in at least one of the two alleles. A typical execution of this job may generate O(1000) output files. Figure 7-*bottom* shows the averaged I/O wait measurements in seconds for I/O read and write operations for the `individuals` jobs. Although there is a slight variation in the time waiting for I/O operations (i.e., makespan is also impacted by the job's computing tasks), a significant performance improvement is seen when using the burst buffer. In summary, burst buffers not only improve I/O performance of read and write operations, but also mitigates the cost of handling a very large number of files.

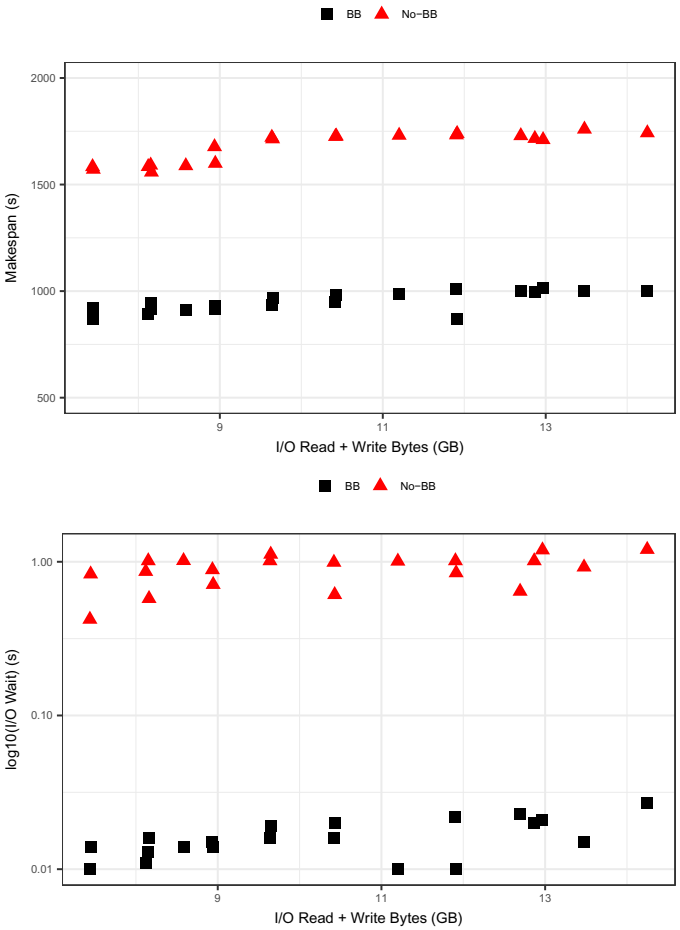

Figure 7: Average makespan (top) and I/O wait (bottom) in seconds for each `individuals` job performing I/O operations on the PFS (red triangles) or the BB (black squares).

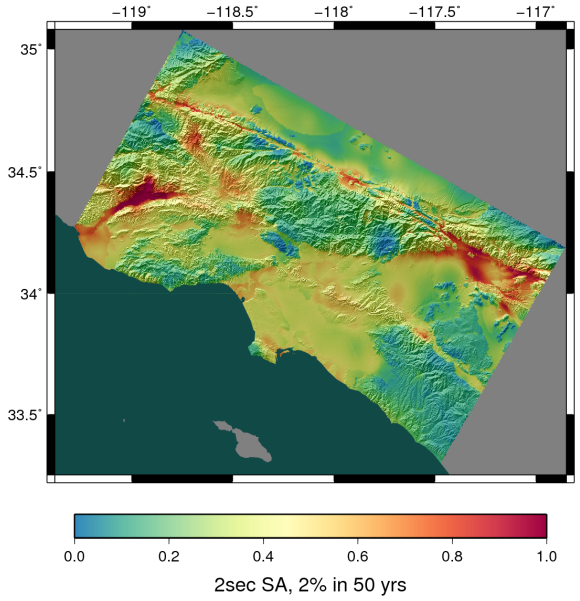

Figure 8: CyberShake hazard map for Southern California, showing the spectral accelerations at a 2-second period exceeded with a probability of 2% in 50 years.

4.3. Burst Buffers for High-Performance Workflows

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 execution of a large-scale high-performance workflow.

4.3.1. The CyberShake Workflow

As part of its research program of earthquake system science, the Southern California Earthquake Center (SCEC) [38] has developed CyberShake [39], 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 [40], 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 8).

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 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) [41]. 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.

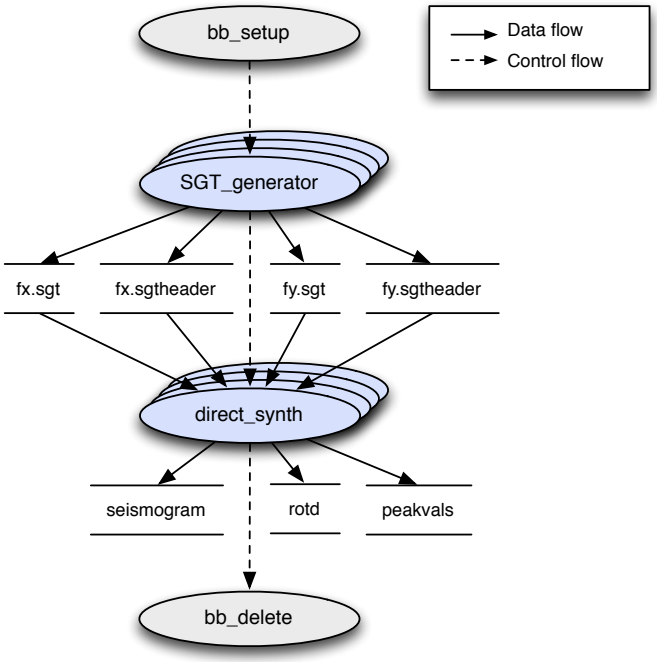

Figure 9: A general representation of the CyberShake test workflow.

4.3.2. Workflow Implementation

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[3]. Figure 9 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.

Following our workflow model described in Section 3.1 (Figure 3), the `SGT_generator` job would only start to run once the `bb_setup` job has created the BB reservation. By using DAGMan's `PRE script` concept, we removed the control dependency between BB creation and the `SGT_generator` job, which has a PRE script that checks whether the BB reservation is up and running.

4.3.3. Experiment Conditions

For the experiments conducted in this section, 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 reduces the number of CPU cycles needed. The `direct_synth` job remains the same.

As in our model we create a single BB persistent reservation per workflow run, it is crucial that the I/O throughput obtained with the BB overcomes the application I/O bottleneck. We performed several runs of the CyberShake test workflow (Figure 9) with different numbers of computational nodes (1–313), and with different numbers of rupture files (1–5,734) 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 increases in this scenario, the complexity of the computation is significantly augmented as the number of rupture files increases (i.e., the increase in 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% (averaged from 10 workflow runs).

4.3.4. Results and Discussion

Overall Write Operations. Figure 10-*top* shows the average I/O performance measurement 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 10-*bottom*) represent the average runtime gain for "I/O write" op-

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

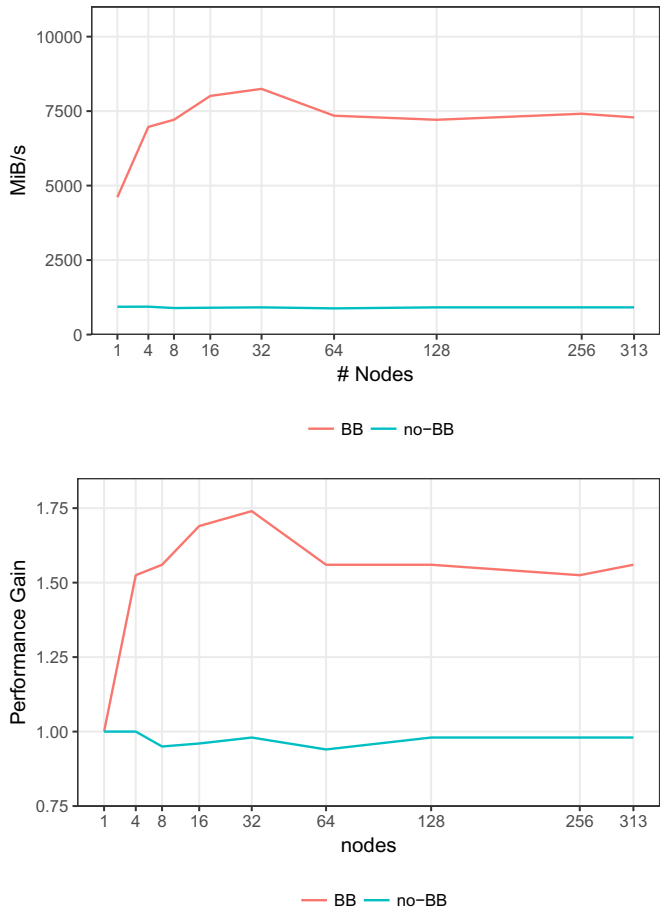

Figure 10: Average I/O performance estimate for write operations at the MPI-IO layer (top), and average I/O write performance gain (bottom) for the `SGT_generator` job.

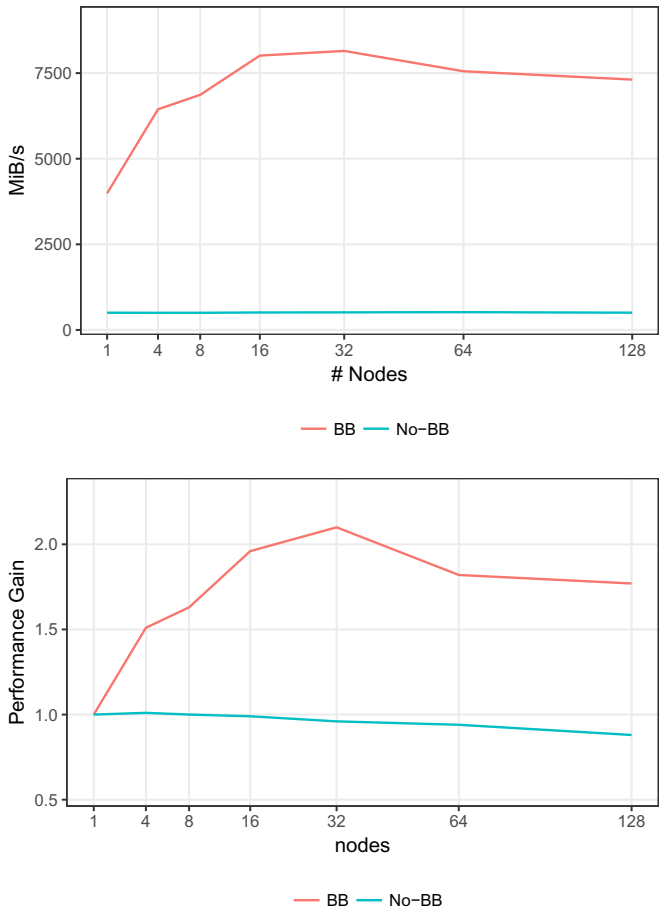

Figure 11: I/O performance estimate for read operations at the MPI-IO layer (top), and average I/O write performance gain (bottom) for the `direct_synth` job.

erations (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). When we increase 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 of the burst buffers, the performance degradation is below 10% and the job runtime significantly benefits from the high degree of parallelism.

Overall Read Operations. Figure 11-*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 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 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 read performance gain values (Figure 11-*bottom*) are marginally higher. This result is due to the lower performance in 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 queuing 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

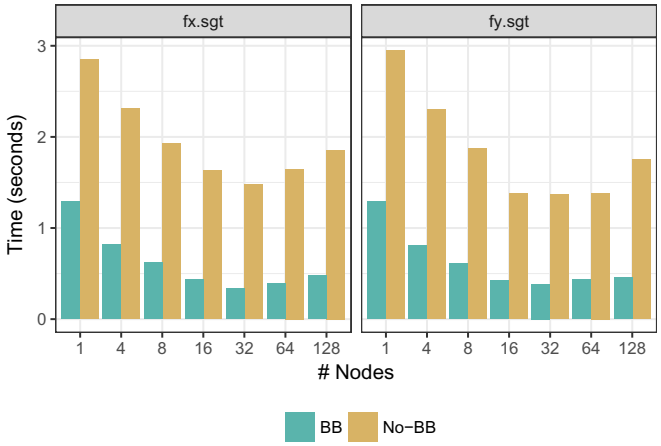

Figure 12: POSIX module data: Average time consumed in I/O read operations per process for the `direct_synth` job.

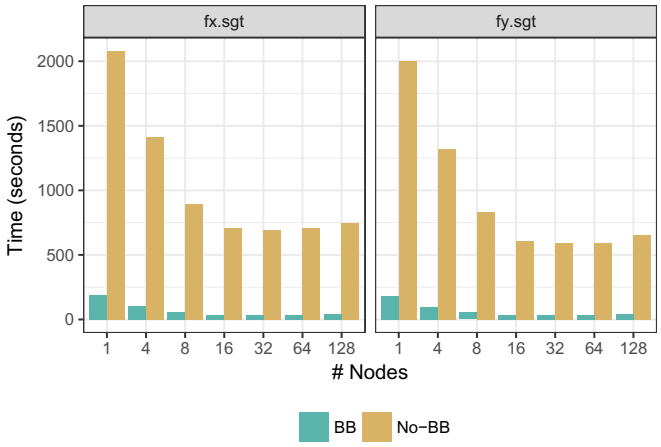

Figure 13: MPI-IO module data: Average time consumed in I/O read operations per process for the `direct_synth` job.

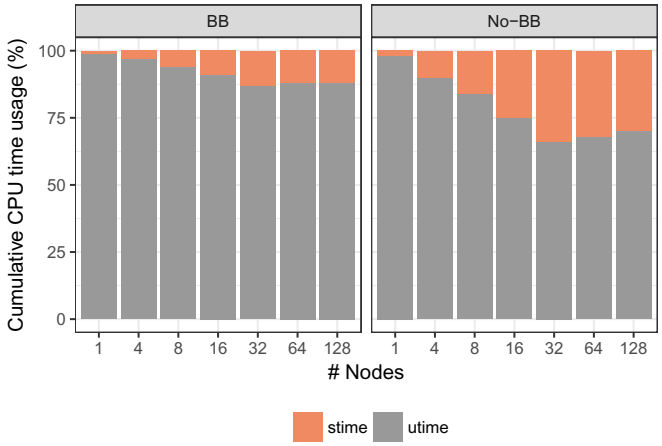

Figure 14: Ratio between the cumulative time spent in the user (`utime`) and kernel (`stime`) spaces for the `direct_synth` job, for different numbers of nodes.

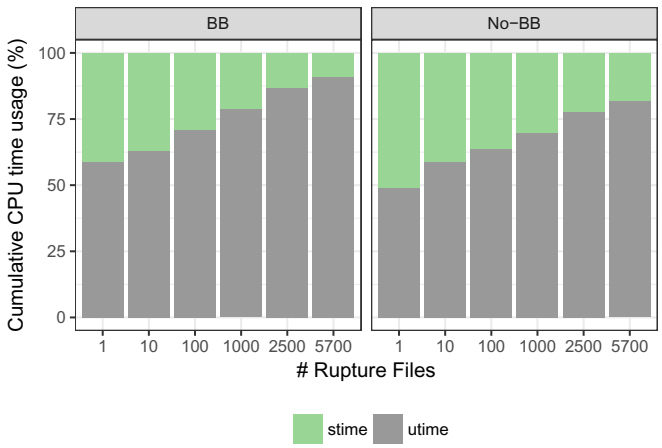

Figure 15: Ratio between the cumulative time spent in the user (`utime`) and kernel (`stime`) spaces for the `direct_synth` job for different numbers of rupture files.

BB to improve the efficiency of other users' jobs on the system. Therefore, if the queuing 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 12 and 13 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 12) represent buffering and synchronization operations with the system. Thus, although there is a visible difference in the average time spent 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 13 shows the average effective time spent per process performing MPI-IO operations. As expected, the average time spent in I/O read operations decreases as more process are used. Note that for larger configurations ($\geq$ 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 decline observed in Figure 11. Workflow executions using the BB accelerate I/O read operations up to 10 times in average. 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 for I/O operations and processing in user space.

Cumulative CPU time. Figure 14 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.

Job	Avg. Makespan (h)	
	PFS	BB
AWP-ODC-SGT	1.09	0.95
direct_synth	8.13	6.97

Table 1: Average job turnaround time in hours (makespan) for the `AWP-ODC-SGT` and `direct_synth` jobs performing I/O operations to the PFS or the BB. `AWP-ODC-SGT` runs used 313 nodes, while `direct_synth` runs used 64 nodes.

Rupture Files. As described in the beginning of Section 4.3, 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 15 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 drives most of the CPU (user space) activities for the `direct_synth` job. 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.

Table 1 summarizes the average makespan for both the `AWP-ODC-SGT` (313 nodes) and `direct_synth` jobs (64 nodes). For the first job, the use of BB improves the job execution time by about 11%, and for the second job by about 15%. Overall, the absolute workflow makespan (i.e., queuing time is ignored) is improved of about 13% in average.

5. Related Work

Efficient workflow execution and scheduling are an extensively researched topic within the field of scientific workflows. A plethora of studies have targeted 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 [42, 27, 43, 44]. Although these solutions may improve workflow execution efficiency, hardware and system 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 [45], 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. The technique focused on data flow and network adaptation. However, such approaches are not applicable 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 [11, 12, 10]. For instance, in [12] an empirical evaluation of a BB implementation with an I/O-bound benchmark application resulted in a speedup factor of 20 when compared to the PFS only solution (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 [8] is a high-performance burst buffer system on top of Memcached [46], 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 [10] 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. [11] 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 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. Use of BB for improving I/O throughput in a workflow is presented in [17]. The runtime of the post-processing step of the scripted workflow is improved up to a factor of 4. A limitation of the work is that changes to the application code were necessary.

The pioneer work on workflow performance characterization using burst buffers was presented in [15], where two workflow applications from LBNL running on Cori were evaluated. The work was a first step towards the efficient use of BB for scientific workflows using a methodology composed of workloads, resources, performance tools, and workload configuration. Our contributions in this paper advances this previous work in the following ways: (1) we evaluate two large data-intensive real-world workflows (consumes/generates over 550 GB of data)—a high-throughput multi-threaded workflow, and a high-performance workflow; (2) we compare the performance gain for using a BB and identify its limitations for the evaluated applications; and (3) we measure the impact of BB at different levels of application parallelism. Although both papers target optimizing the use of BB for scientific workflows, we focus on the mechanisms to enable the seamless use of BB for the execution of scientific workflows—in particular when running legacy applications. The MaDaTS [16] framework provides full-fledged management of the data lifecycle for workflow execution on HPC systems. Similarly to this work, scientific worklfow applications (both real and synthetic) are evaluated on Cori. While this related work targets scientific applications that handle small volumes of data and large number of compute jobs, our work targets workflows composed by a small

number of computing jobs (though parallel applications requiring many cores) producing/consuming large volumes of data. We therefore argue that the findings and conclusions previously drawn by the community and the ones presented in this paper constitute a solid basis knowledge from enabling the efficient support of burst buffers in workflow systems.

6. Conclusions

In this paper, we explored the impact of burst buffers on the performance of two real-world large-scale data-intensive scientific workflow applications the 1000 genome workflow, and SCEC CyberShake. Using a software stack including Pegasus-WMS and HTCondor, we ran the workflows on the Cori system at NERSC. The workflows included provisioning and releasing remote-shared BB nodes. We found that for our genomics high-throughput multi-threaded application, which read about 11 GB per job and wrote to O(1000) files, overall workflow makespan was improved by a factor of 2. For our Earth science high-performance application, which wrote and read about 550 GB of data, the I/O write performance was improved by a factor of 9, and the 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.

To conduct the above experiment, we have presented an architectural model for seamlessly enabling burst buffers support in workflow management systems, in particular for legacy applications. We have also described our initial implementation of the proposed model with Pegasus-WMS. Our approach used a persistent BB reservation for staging workflow files during execution, however typical queue latency in HPC systems may lead to poor performance. Therefore, we plan to investigate hybrid scheduling approaches where persistent and temporary reservations are used interchangeably throughout the workflow execution.

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 [3, 4], as well as consider more intrusive approaches for changing workflow applications and systems to optimize for burst buffer usage. Future work also includes a full-fledged reference implementation of a production solution for workflow systems, in particular Pegasus, to include all 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 1000 genome and CyberShake workflows (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", by NSF contract number #1664162, " SI2-SSI: Pegasus: Automating Compute and Data Intensive Science", and by NSF contract number #1741040, "BIGDATA: IA: Collaborative Research: In Situ Data Analytics for Next Generation Molecular Dynamics Workflows". 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] I. J. Taylor, E. Deelman, D. B. Gannon, M. Shields, Workflows for e-Science: scientific workflows for grids, Springer Publishing Company, Incorporated, 2007.

[2] C. S. Liew, M. P. Atkinson, M. Galea, T. F. Ang, P. Martin, J. I. V. Hemert, Scientific workflows: moving across paradigms, ACM Computing Surveys (CSUR) 49 (4) (2016) 66.

[3] T. Johnston, B. Zhang, A. Liwo, S. Crivelli, M. Taufer, In situ data analytics and indexing of protein trajectories, Journal of computational chemistry 38 (16) (2017) 1419–1430.

[4] M. Dreher, B. Raffin, 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, 2014, pp. 277–286.

[5] https://www.nitrd.gov/nsci/files/NSCI2015WorkshopReport06142016.pdf (2015).

[6] Summit, `https://www.olcf.ornl.gov/summit/`.

[7] Aurora, `https://aurora.alcf.anl.gov`.

[8] T. Wang, S. Oral, Y. Wang, B. Settlemyer, S. Atchley, W. Yu, Burstmem: A high-performance burst buffer system for scientific applications, in: Big Data (Big Data), 2014 IEEE International Conference on, IEEE, 2014, pp. 71–79.

[9] W. Bhimji, D. Bard, M. Romanus, D. Paul, A. Ovsyannikov, B. Friesen, M. Bryson, J. Correa, G. K. Lockwood, V. Tsulaia, et al., Accelerating science with the nersc burst buffer early user program, CUG2016 Proceedings.

[10] T. Wang, K. Mohror, A. Moody, K. Sato, W. Yu, An ephemeral burst-buffer file system for scientific applications, in: High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for, IEEE, 2016, pp. 807–818.

[11] S. Herbein, D. H. Ahn, D. Lipari, T. R. Scogland, M. Stearman, M. Grondona, J. Garlick, B. Springmeyer, M. Taufer, 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, 2016, pp. 69–80.

[12] W. Schenck, S. El Sayed, M. Foszczynski, W. Homberg, D. Pleiter, Evaluation and performance modeling of a burst buffer solution, ACM SIGOPS Operating Systems Review 50 (1) (2017) 12–26.

[13] G. S. Markomanolis, B. Hadri, R. Khurram, S. Feki, Scientific applications performance evaluation on burst buffer, in: International Conference on High Performance Computing, Springer, 2017, pp. 701–711.

[14] W. Bhimji, D. Bard, K. Burleigh, C. Daley, S. Farrell, M. Fasel, B. Friesen, L. Gerhardt, J. Liu, P. Nugent, et al., Extreme i/o on hpc for hep using the burst buffer at nersc, in: Journal of Physics: Conference Series, Vol. 898, IOP Publishing, 2017, p. 082015. doi:10.1088/1742-6596/898/8/082015.

[15] C. S. Daley, D. Ghoshal, G. K. Lockwood, S. Dosanjh, L. Ramakrishnan, N. J. Wright, Performance characterization of scientific workflows for the optimal use of burst buffers, Future Generation Computer Systemsdoi:10.1016/j.future.2017.12.022.

[16] D. Ghoshal, L. Ramakrishnan, Madats: Managing data on tiered storage for scientific workflows, in: Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing, ACM, 2017, pp. 41–52. doi:10.1145/3078597.3078611.

[17] A. Ovsyannikov, M. Romanus, B. Van Straalen, G. H. Weber, D. Trebotich, Scientific workflows at datawarp-speed: accelerated data-intensive science using nersc's burst buffer, in: Parallel Data Storage and data Intensive Scalable Computing Systems (PDSW-DISCS), 2016 1st Joint International Workshop on, IEEE, 2016, pp. 1–6. doi:10.1109/PDSW-DISCS.2016.005.

[18] R. Ferreira da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, E. Deelman, A characterization of workflow management systems for extreme-scale applications, Future Generation Computer Systems 75 (2017) 228–238. doi:10.1016/j.future.2017.02.026.

[19] http://www.nersc.gov/users/computational-systems/cori/ (2018).

[20] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, K. Wenger, Pegasus: a workflow management system for science automation, Future Generation Computer Systems 46 (2015) 17–35. doi:10.1016/j.future.2014.10.008.

[21] R. Ferreira da Silva, S. Callaghan, E. Deelman, On the use of burst buffers for accelerating data-intensive scientific workflows, in: 12th Workshop on Workflows in Support of Large-Scale Science (WORKS'17), 2017, pp. 2:1–2:9. doi:10.1145/3150994.3151000.

[22] Open science grid, `https://opensciencegrid.org`.

[23] Xsede, `https://www.xsede.org`.

[24] R. Ferreira da Silva, E. Deelman, R. Filgueira, K. Vahi, M. Rynge, R. Mayani, B. Mayer, Automating environmental computing applications with scientific workflows, in: Environmental Computing Workshop (ECW'16), IEEE 12th International Conference on e-Science, 2016, pp. 400–406. doi:10.1109/eScience.2016.7870926.

[25] R. Ferreira da Silva, R. Filgueira, E. Deelman, E. Pairo-Castineira, I. M. Overton, M. Atkinson, Using simple pid controllers to prevent and mitigate faults in scientific workflows, in: 11th Workflows in Support of Large-Scale Science (WORKS'16), 2016, pp. 15–24.

[26] R. Ferreira da Silva, R. Filgueira, E. Deelman, E. Pairo-Castineira, I. M. Overton, M. Atkinson, Using simple pid-inspired controllers for online resilient resource management of distributed scientific workflows, Future Generation Computer Systems 95 (2019) 615–628. doi:10.1016/j.future.2019.01.015.

[27] J. Liu, E. Pacitti, P. Valduriez, M. Mattoso, A survey of data-intensive scientific workflow management, Journal of Grid Computing 13 (4) (2015) 457–493.

[28] T. Wang, Exploring novel burst buffer management on extreme-scale hpc systems, Ph.D. thesis, The Florida State University (2017).

[29] D. Henseler, B. Landsteiner, D. Petesch, C. Wright, N. J. Wright, Architecture and design of cray datawarp, in: Proc. Cray Users' Group Technical Conference (CUG), 2016.

[30] C. Docan, M. Parashar, S. Klasky, Dataspaces: an interaction and coordination framework for coupled simulation workflows, Cluster Computing 15 (2) (2012) 163–181.

[31] R. Ferreira da Silva, G. Juve, M. Rynge, E. Deelman, M. Livny, Online task resource consumption prediction for scientific workflows, Parallel Processing Letters 25 (3). doi:10.1142/S0129626415410030.

[32] G. Aupy, O. Beaumont, L. Eyraud-Dubois, Sizing and partitioning strategies for burst-buffers to reduce io contention, in: 33rd IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2019.

[33] J. Frey, Condor dagman: Handling inter-job dependencies, `http://www.bo.infn.it/calcolo/condor/dagman` (2002).

[34] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: the condor experience, Concurrency and computation: practice and experience 17 (2-4) (2005) 323–356.

[35] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, R. Ross, Understanding and improving computational science storage access through continuous characterization, ACM Transactions on Storage (TOS) 7 (3) (2011) 8.

[36] 1000 Genomes Project Consortium, et al., A global reference for human genetic variation, Nature 526 (7571) (2012) 68–74.

[37] G. Juve, B. Tovar, R. Ferreira da Silva, D. Król, D. Thain, E. Deelman, W. Allcock, M. Livny, Practical resource monitoring for robust high throughput computing, in: 2nd Workshop on Monitoring and Analysis for High Performance Computing Systems Plus Applications (HPCMASPA'15), 2015, pp. 650–657. doi:10.1109/CLUSTER.2015.115.

[38] Southern california earthquake center, http://www.scec.org (2018).

[39] R. Graves, T. H. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner, et al., Cybershake: A physics-based seismic hazard model for southern california, Pure and Applied Geophysics 168 (3-4) (2011) 367–381.

[40] Working Group on California Earthquake Probabilities, The uniform california earthquake rupture forecast, version 2 (2008).
URL `https://pubs.usgs.gov/of/2007/1437/`

[41] Y. Cui, E. Poyraz, J. Zhou, S. Callaghan, P. Maechling, T. H. Jordan, L. Shih, P. Chen, Accelerating cybershake calculations on xe6/xk7 platforms of blue waters, in: Proceedings of Extreme Scaling Workshop 2013, 2013.

[42] J. Ma, W. Liu, T. Glatard, A classification of file placement and replication methods on grids, Future Generation Computer Systems 29 (6) (2013) 1395–1406.

[43] L. B. Costa, H. Yang, E. Vairavanathan, A. Barros, K. Maheshwari, G. Fedak, D. Katz, M. Wilde, M. Ripeanu, S. Al-Kiswany, The case for workflow-aware storage: An opportunity study, Journal of Grid Computing 13 (1) (2015) 95–113.

[44] L. Liu, M. Zhang, Y. Lin, L. Qin, 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, 2014, pp. 837–846.

[45] A. Mandal, P. Ruth, I. Baldin, Y. Xin, C. Castillo, G. Juve, M. Rynge, E. Deelman, J. Chase, Adapting scientific workflows on networked clouds using proactive introspection, in: IEEE/ACM Utility and Cloud Computing (UCC), 2015. doi:10.1109/UCC.2015.32.

[46] B. Fitzpatrick, Distributed caching with memcached, Linux journal 2004 (124) (2004) 5.