Modeling the Performance of Scientific Workflow Executions on HPC Platforms with Burst Buffers

Loïc Pottier*, Rafael Ferreira da Silva*, Henri Casanova†, Ewa Deelman*
*USC Information Sciences Institute, Marina Del Rey, CA, USA
†University of Hawai'i at Manoa, Honolulu, HI, USA
{lpottier,rafsilva,deelman}@isi.edu, henric@hawaii.edu

Abstract*—Scientific domains ranging from bioinformatics to astronomy and earth science rely on traditional high-performance computing (HPC) codes, often encapsulated in scientific workflows. In contrast to traditional HPC codes that employ a few programming and runtime approaches that are highly optimized for HPC platforms, scientific workflows are not necessarily optimized for these platforms. As an effort to reduce the gap between compute and I/O performance, HPC platforms have adopted intermediate storage layers known as burst buffers. A burst buffer (BB) is a fast storage layer positioned between the global parallel file system and the compute nodes. Two designs currently exist: (i) *shared*, where the BBs are located on dedicated nodes; and (ii) *on-node*, in which each compute node embeds a private BB. In this paper, using accurate simulations and real-world experiments, we study how to best use these new storage layers when executing scientific workflows. These applications are not necessarily optimized to run on HPC systems, and thus can exhibit I/O patterns that differ from that of HPC codes. Thus, we first characterize the I/O behaviors of a real-world workflow under different configuration scenarios on two leadership-class HPC systems (Cori at NERSC and Summit at ORNL). Then, we use these characterizations to calibrate a simulator for workflow executions on HPC systems featuring shared and private BBs. Last, we evaluate our approach against a large I/O-intensive workflow, and we provide insights on the performance levels and the potential limitations of these two BBs architectures.

***Index Terms*—Scientific workflow, Simulations, Burst buffers, High-Performance Computing, Performance Characterization**

I. Introduction

The scientific workflow abstraction is one of the pillars that support large-scale science discoveries [1], [2]. In the past few years, several shifts have been observed, from the perspective of both scientific workflows and high-performance computing (HPC) architectures. Evolving from traditionally compute-intensive and loosely-coupled settings, next-generation scientific workflows are moving towards tightly-coupled and data- and I/O-intensive configurations [3], [4]. At the same time, HPC platforms have begun to leverage new storage technologies such as non-volatile memory and SSDs. For instance, most recent supercomputers include burst buffers nodes [3]. A burst buffer (denoted as BB in this paper) is an intermediate storage layer between the compute nodes and the long-term storage [5] – primarily designed to improve the performance of the checkpoint/restart resiliency protocol used in HPC codes by reducing the pressure on the parallel file system (denoted as PFS in this paper). Many studies have focused on improving the performance of checkpointing phases via multi-level checkpointing [6], [7], or via file systems optimizations for BBs [8].

BBs have also been used to improve the I/O performance of scientific applications. With the advent of Big Data and extreme scale computing, data-intensive workflows process/generate large volumes of data, making I/O the key element hindering workflow execution performance. Consequently, recent works [9], [10] have conducted preliminary studies on how the use of BBs could mitigate I/O overheads of "staging in/out" (intermediate) workflow data in HPC systems. Although these studies have demonstrated substantial performance gain when compared to the traditional PFS approach, there is still a need for conducting an in-depth analysis of BBs capabilities regarding different workflow structures and task requirements – some tasks may generate small numbers of very large files, while others may generate large numbers of very small files. Such analysis may unveil limitations of current BB solutions for supporting current and emerging high-profile applications, such as scientific workflows, due to their I/O needs and task-dependency structures.

The objective of this work is twofold: (i) understand the behavior and impact of using new deep-memory hierarchies such as BBs to accelerate the execution of scientific workflows; and (ii) leverage this understanding to implement and calibrate a simulator framework that can in turn be used to explore how different BB architectures impact workflow execution performance.

To achieve the above we conduct a comprehensive analysis of the impact of BBs on the performance, and performance variability, of a real-world production workflow on two leadership-class HPC systems (Cori at NERSC [11], and Summit at ORNL [12]). We leverage the gathered performance data to develop a performance model, which in turns makes it possible to evaluate different BB configurations in simulation. Several studies [13]–[15] have focused on developing efficient data placement schemes and performance models targeting dual memory systems (i.e., systems with one large slow memory and one small fast memory). Unfortunately, the design space for heuristics that optimize the data placement between PFS and BB is enormous. As a result, it cannot be explored effectively by relying on time- and energy-consuming experiments conducted on real platforms. As an alternative, our proposed lightweight simulation approach, which achieves a great trade-off between accuracy and usability, makes it possible to explore this design space thoroughly and quickly. It thus provides the necessary foundation for developing efficient data placements heuristics for optimizing workflow execution on BB-equipped platforms.

Specifically, this work makes the following contributions:

1) Collection of performance data from a real-world scientific workflow executions on two leadership-class HPC systems equipped with two different BBs designs;
2) An in-depth analysis of the workflow performance regarding resource allocation and BBs configurations;
3) An application model for I/O-intensive workflows that accounts for the use of BBs on HPC systems – along with an experimental evaluation of the proposed model under different configuration scenarios;
4) A simulation framework for experimental evaluation of potential future designs of, or data placement solutions enabled by burst buffers systems; and
5) Extensive evaluation of the simulation framework using a large-scale I/O-intensive bioinformatics workflow, and comparison of simulation results with actual experiments performed in previous work.

This paper is organized as follows. Section II reviews the related research. Section III presents collected experimental data for a real-world use case in two leadership-class HPC systems. Section IV describes our model and simulation framework, and assesses their accuracy using the collected experimental data. Finally, Section V concludes with a brief summary of results and a discussion of future research directions.

II. Background and Related Work

The gap between computational and I/O performance in current HPC systems remains critical [3]. To reduce this gap, many HPC centers have adopted a fast intermediate storage layer called burst buffers (BBs) [5]. The BB concept was first developed to improve checkpointing performance, i.e., to alleviate the I/O pressure created by frequent checkpoints [16]. Several studies [6], [17], [18] have explored using BBs to improve checkpointing performance. For instance, BurstMem [19] is a log-structured BB system build on top of Memcached, a distributed caching system for optimizing I/O access patterns, e.g., checkpoint/restart or data staging. GekkoFS [20] is a temporary, highly scalable parallel file system (PFS) specifically optimized to scale metadata operations for accesses to a single directory or even to a single file, which are known to not scale well on traditional PFSs.

Currently, two leading implementations of BBs are competing in the HPC landscape. The first considers on-node local BBs – each compute node has its own local BB. The second considers dedicated BB nodes where all compute nodes can store data. Some studies have attempted to determine architectural advantages and limitations of each approach regarding classic HPC I/O behaviors [21], [22]. However, these works target simplistic scenarios restricted to a few workloads that do not necessarily expose the relevant trade-offs inherent to the different BBs implementations.

Recent studies [9], [10], [16], [23], [24] have demonstrated the utility of BBs for accelerating scientific workflows by reducing costly disk I/O. These studies show substantial performance improvements over the traditional PFS scenario. The evaluation of the I/O performance of two data analytics workflows executing on a leadership-class supercomputer (Cori at NERSC) showed that there is significant performance variation under different I/O patterns conditions [24]. Specifically, by distinguishing data and metadata operations, it is demonstrated that a sequential workflow pipeline does not use enough I/O parallelism to saturate the BB bandwidth. It is also noted that workflows using database back-ends and Python scripts are often limited by metadata performance. Data management for scientific workflows has also been studied. In particular, MaDaTS [25] is a software architecture for managing data used by scientific workflows on recent HPC architectures that feature multiple storage layers. MaDaTS proposes a virtual data space that acts as an intermediate storage representation to hide the complexity of the physical storage system to the workflow execution.

On the modeling and simulation aspect, the CODES simulation framework [26] has been extensively used to model BB performance [16], [27]–[29]. CODES is a parallel discrete-event simulation that accurately aims to model the network topology, the I/O stacks, and the storage layers. Few theoretical models have been proposed. In [30] the authors employ a probabilistic model to capture the behavior of periodic HPC application running on a shared BB architecture. Simulation has also been used in a provisioning system to provide accurate, multi-tenant simulations that model realistic application and storage workloads from HPC systems [31]. Although these proposed models yield fairly accurate estimates, they do not consider task-dependency structures, as present in scientific workflows, in which data stage-in between dependent tasks may severely impact overall execution performance.

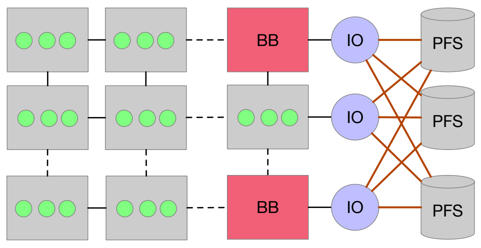
III. Experimental Evaluation

In this section, we perform experiments to measure the actual performance of I/O read and write operations from/to BBs for processing the I/O-bound tasks of a real-world data-intensive workflow.

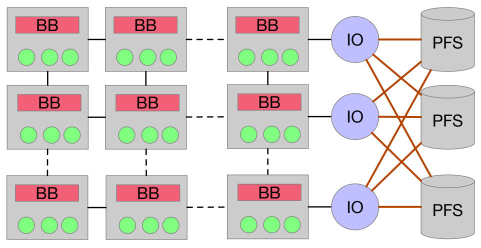
A. Experimental Platform

Although BBs have become popular in supercomputers, determining the optimal BB architecture is still an open question. Currently, two main representative architectures have been deployed: (i) shared burst buffers (remote-shared BB), and (ii) on-node burst buffers (node-local BB) [32]. Below, we briefly describe these architectures and underline their differences.

1) Shared burst buffers: The Cori system, at the National Energy Research Scientific Computing Center (NERSC), provides remote-shared, allocatable BBs. Cori is a Cray XC40 which delivers about 30 PetaFlops using 2,388 Intel Haswell



(a) Cori architecture.



(b) Summit architecture.

Figure 1. BB architectures on two leadership-class HPC systems. Burst Buffers are in red, processing units in green, and I/O nodes in blue.

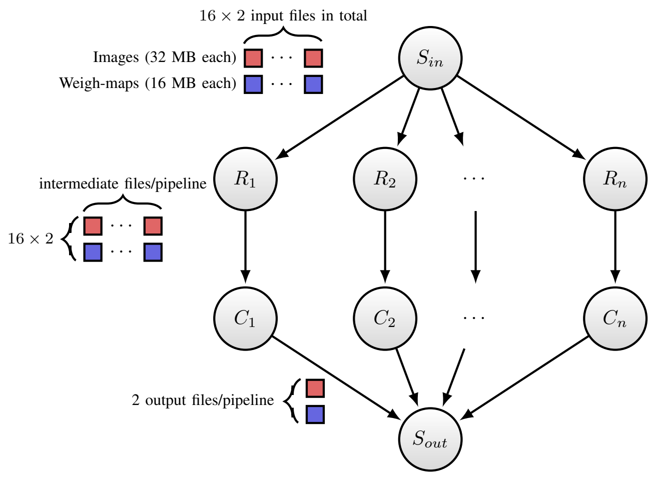


Figure 2. Overview of the SWarp workflow composed of a data *stage-in* task S_{in}, and several pipelines composed of *Resample* and *Combine* tasks (respectively, R_i and C_i tasks). The 16 input images are shown in red, and the 16 corresponding weight maps are shown in blue.

nodes and 9,688 Intel Xeon Phi KNL nodes connected through a Cray Aries interconnect. The BB resides on specialized nodes that bridge the internal interconnect of the compute system and the storage area network (SAN) fabric of the storage system through the I/O nodes. Each BB node hosts 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 BB node provides approximately 6.4 TB of usable capacity and a peak of approximately 6.5 GB/sec of sequential read and write bandwidth. In this paper, we only run experiments using the Haswell nodes. The architecture is depicted in Figure 1(a). NERSC's burst buffer implementation provides two performance tuning modes: *striped* and *private*. The former distributes files over multiple BB nodes, while in the latter each compute node gets its own namespace, which potentially affords improved performance for metadata handling. In this paper, we investigate both performance tuning modes.

2) On-node burst buffers: The Summit system, at the Oak Ridge National Laboratory (ORNL), provides node-local, allocatableBBs. Summit is an IBM Power System AC922 which delivers about 200 PetaFlops using 4,608 nodes, with two POWER9 and six Nvidia Volta V100s per node, connected through a dual-rail Mellanox EDR InfiniBand interconnect. The BB resides on each compute node and are equipped with 1.6 TB Samsung PM1725a NVMe solid state drivers, which in total amount to 7.3 PB. Each local BB has an expected peak write performance of 2.1 GB/s, and an approximately peak read performance of 6 GB/s. The architecture is depicted in Figure 1(b).

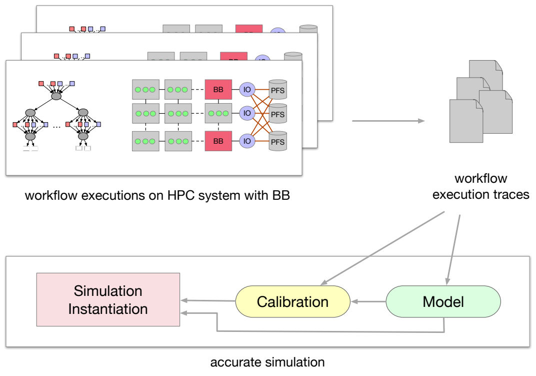
B. Scientific Application

The SWarp cosmology workflow [24] is used in large-scale sky surveys for combining overlapping raw images of the night sky into high quality reference images. The workflow is composed of thousands of embarrassingly parallel SWarp pipelines, in which each pipeline produces an image for a predefined resolution of the sky. For this experimental evaluation, we consider an instance of the SWarp workflow in which average input to each workflow pipeline has 16 input images (32 MiB each) and 16 input weight maps (16 MiB each). SWarp is written in C and multithreaded with POSIX threads. The multithreading strategy involves different threads operating on different regions of the same image.

Figure 2 shows an overview of the workflow structure. The entry task is a *stage-in* task, after which a number of independent pipelines must be executed. Although each pipeline is composed of two sequential compute tasks, we argue that these tasks and the combined pipelines can be used as a proxy application to estimate I/O performance of scientific workflows. This is mainly due to the multithreading characteristics of the tasks (note that several similar tasks belonging to different pipelines will run in parallel), which in turn increases the number of concurrent I/O operations at runtime. While the I/O pattern of individual tasks is simple, the complex workflow structure allow us to evaluate a number of key experimental scenarios (discussed hereafter), and therefore draw conclusions that can be related to most patterns that commonly occur in production scientific workflows.

C. Experimental Scenario

Figure 3 shows an overview of our experimental approach. We first perform real-world executions of SWarp on Cori and Summit when varying the number of cores, the number of pipelines that run in parallel, and the proportion of files read/written from/to the BB. We then investigate the performance impact of using each of the two different BBs modes (private and striped, as proposed by Cray's implementation of the shared BB architecture used in Cori), as well as the



BB
IO
PFS
workflow executions on HPC system with BB
workflow execution traces
Simulation Instantiation
Calibration
Model
accurate simulation

Figure 3. Overview of the experimental scenario from real experiments to model and simulations.

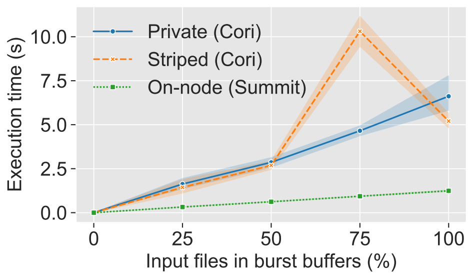


Figure 4. Execution time of the Stage-In task of the SWarp workflow with one pipeline and 32 cores per tasks, vs. the percentage of input files stored in BBs.

two representative architectures implementations (Cori and Summit). This is depicted in the top part of Figure 3.

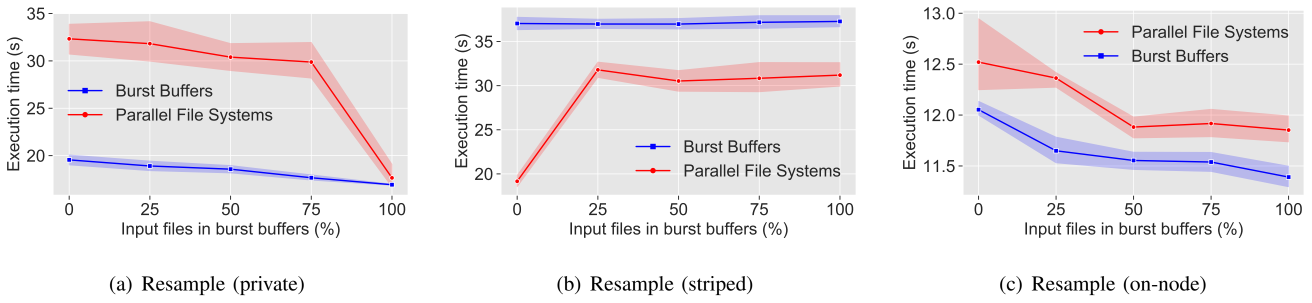
Unfortunately, it is not feasible to explore experimentally all relevant scenarios at scale via real-world executions on Cori and Summit. This is because these systems are shared with other users, which limits the number and scale of experiments that can be feasibly conducted. But it is also because it is desirable to explore scenarios for platform configurations that differ from that of the Cori and Summit systems. As a result, we use experimental data obtained on these systems, as described hereafter, to calibrate (and validate) a simulator for workflow executions on HPC platforms equipped with BBs. This is depicted in the bottom part of Figure 3. The simulator is a software artifact that models the functional and performance behaviors of software and hardware stacks of interest for workflow executions, and thus allows us to explore storage management options at scale for arbitrary platform and workflow configurations. The simulator is described in Section IV.

D. Experimental Results

In this sub-section, we explore the parameter space of workflow executions on Cori and Summit HPC systems. We aim to characterize the behavior of the two representative implementations, as well as the different modes (private and striped) in the shared approach, so that we can derive a model for enabling accurate simulation of scientific workflows on HPC systems with burst buffers. To ensure statistically valid results, each run is averaged over 15 executions on a dedicated single compute node. As I/O measurements are extremely sensitive to background load and both systems are heavily used, we reduce potential interference via the usage of Slurm (Cori) and LSF (Summit) directives – we insure no other jobs are running concurrently on the same node and that symmetric multi-threading (SMT) is deactivated. Note that, in this work, the stage-in task is always sequential.

Impact of data staging into burst buffers. Although BBs have been known to significantly improve the performance of workflow applications [10], [24], there is a perceived cost for staging workflow input data into the intermediate storage layer. Figure 4 shows a comparison of the execution performance (in terms of executed time in seconds) of the Stage-In task of the SWarp workflow – including both representative implementations, and the two modes (private and striped) for the shared approach. For this experiment, we vary the number of workflow input files staged into the BB. As expected, Stage-In execution time increases linearly with the amount of data transferred. The on-node implementation (Summit) outperforms the shared implementation (Cori) up to a factor of 5, which is mostly due to the latency experienced for the remote connection. Notice that both private and striped modes present substantial variations on the measurements (ass seen in the curve envelopes). This variation is due to the variations in competing load on the system (since BBs are shared across user jobs). Also notice an unexpected (poor) behavior when 75% of the input files are staged into the BB when using the striped mode. In spite of our best efforts we were not able to pinpoint the reason for this behavior, which is reproducible. Our best guess is that this behavior may be due to a particular threshold defined in the system configuration that triggers a different mode of operation, which in turn leads to improved performance as the percentage of files stored in BBs further increases. Providing a definite explanation would require an experimental investigation of the system configuration parameters, which can only be done by platform administrators.

On the importance of the BB mode. The implementation of a shared BB design is complex, and must address the challenges of consistency, coherency, and performance. As there are far more compute nodes than I/O and BB nodes, a given BB allocation is usually spread over multiple BB nodes to ensure satisfactory access time. Note that files can also be striped over multiple BBs. As a result, performance in such systems is hard to predict. Recall that Cori provides support to two modes of BB implementations: private and striped (also called shared). In the private mode, access to files in the BB are limited to the compute node (CN) that created them, while in the striped mode files can be accessed by any CN participating in the computation. Figure 5 shows the performance impact for a single pipeline (each task running on 32 cores) when using the two modes. For each scenario, we assess the performance gain


(a) Resample (private) (b) Resample (striped) (c) Resample (on-node)

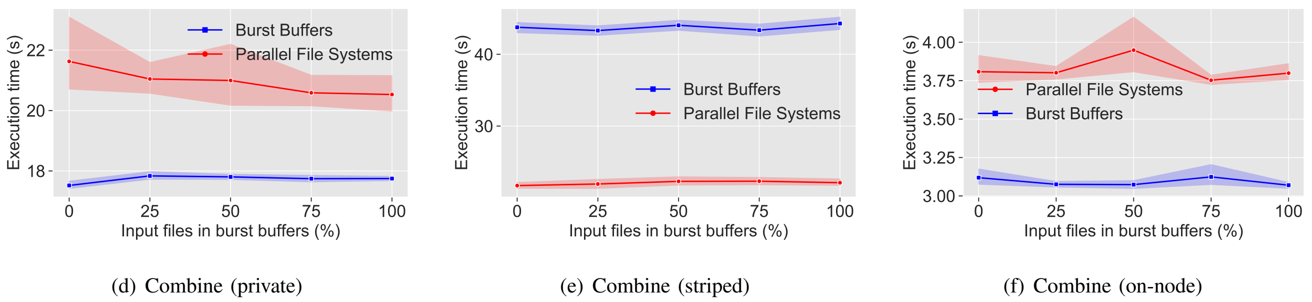
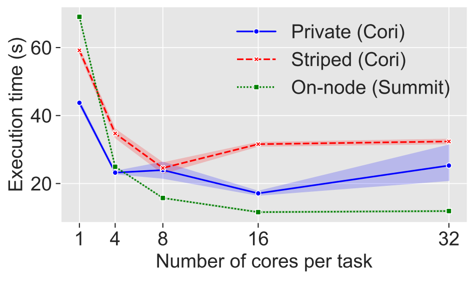

(d) Combine (private) (e) Combine (striped) (f) Combine (on-node)

Figure 5. Execution performance of the Resample and Combine tasks of the SWarp workflow with one pipeline and 32 cores per tasks. Storage layers (BB or PFS) indicate where all intermediate files written by Resample and read by Combine tasks are stored.

when intermediate files written by Resample and consumed by Combine tasks (Figure 2) are entirely stored into the BB or the PFS. While Resample tasks are impacted by both I/O read operations of input files and I/O write operations of intermediate files (in this case either written to the BB of PFS), Combine tasks are solely impacted by I/O read operations.

In Figure 5(a), read operations mostly impact the performance of the Resample tasks in the private mode – as more input files are stored in the BB nodes, the better the performance. Clearly, writing output files to BB nodes is far more efficient than to the PFS (up to a factor of 1.5). A similar trend is observed for the Combine tasks in the private mode (Figure 5(d)), though the execution performance is nearly constant since all read operations are performed on a single storage layer. When compared to the striped mode (Figures 5(b) and 5(e)), the private mode outperforms it by up to two orders of magnitude. Nevertheless, we observe a counter-intuitive result with the striped mode, where the performance slightly decreases when most of the workflow data resides in the BB. Additionally, performing read operations from the PFS yields better performance than from the BB nodes. This is mostly due to the fact that SWarp I/O pattern is $1 : N$ where one task accesses many files, while the striped mode is optimized for a $N : 1$ communication pattern, where many tasks access the same shared file. Therefore, this confirms the well-established fact that understanding the application's communication pattern is critical for effective use of BBs.

For the on-node implementation (Summit, Figures 5(c) and 5(f)), I/O performance increases for both Resample and Combine tasks, with slightly better performance when larger volumes of data are stored in the local BB. In addition, the on-node implementation outperforms the shared implementation up to three orders of magnitude. Notice that the performance of the PFS in Summit yields fairly good performance for the studied workflow (slowdown as low as 0.2). We argue then that data movement between local BBs (e.g., when using more


(a) Resample

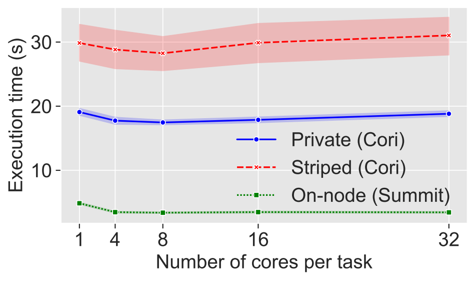
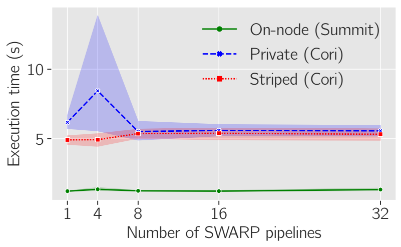
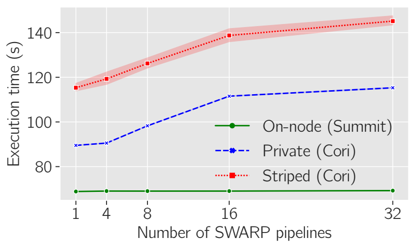

(b) Combine

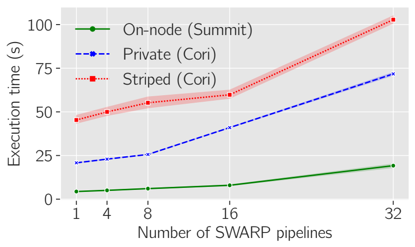
Figure 6. SWarp workflow with one pipeline and 32 threads per tasks and all input files staged into burst buffers.



(a) Stage-In



(b) Resample



(c) Combine

Figure 7. Execution performance of the SWarp workflow when varying the number of pipeline running on one compute node (1 core per pipeline). All file (input and intermediate) are stored into burst buffers.

than a single node) would not significantly slow down the application execution. This result indicates that the on-node implementation would likely scale well for large-scale workflow applications (which we demonstrate in Section IV-C).

Impact of the number of cores per task. Figure 6 shows the performance impact when varying the number of cores. Not surprisingly, Resample benefits from parallelism up to 8 cores in the shared implementation, then performance slightly degrades as the number of cores increases (in particular for the striped mode). For the on-node implementation, the plateau for performance improvement is reached at 16 cores. By contrast, Combine does not benefit from increased parallelism. Combine reads all input files at once and combines them into a singe larger file – this operation involves synchronization and locks; Resample, instead, produces for each input file a corresponding output file in parallel. This result indicates that the performance gap between striped and private modes, and between shared and on-node representative implementations are not in general correlated with the number of cores.

Impact of the number of pipelines. The SWarp workflow is composed of many parallel independent pipelines (Figure 2), thus we study the performance impact when varying the number of pipelines running concurrently in a single compute node (Figure 7). As compute nodes in Cori have 32 cores each, we ran workflows composed of 1 to 32 pipelines (each workflow runs on a single core). As the number of pipelines increases, Resample and Combine tasks are slowed down by up to a factor of 3 in Cori (regardless of the BB allocation mode). As each workflow is only composed of a single Stage-In task, the impact of concurrent access to the BB nodes is minimal. Theoretically, if we ignore sharing interference, the makespan of a SWarp instance with a single pipeline should be nearly the same as an instance with 32 pipelines – as each pipeline is independent and the BB bandwidth is typically significantly larger than the PFS bandwidth (up to 5×). And yet, the results show a measurable slowdown as the number of concurrent pipelines increases. This, surprisingly, indicates that the contention for the BB bandwidth plays an important role in the workflow performance. In other words, the effective bandwidth achieved by this workflow implementation is well below the peak bandwidth on the Cori system. This is likely due to the fact that this implementation uses standard POSIX I/O operations, which are not known for being particularly efficiency (unlike, e.g., MPI-IO operations used by HPC codes). Notice that the performance decrease as the number of pipelines increases on the on-node implementation is nearly negligible for the Stage-In and Resample tasks, while a more significant performance decrease is observed for the Combine tasks.

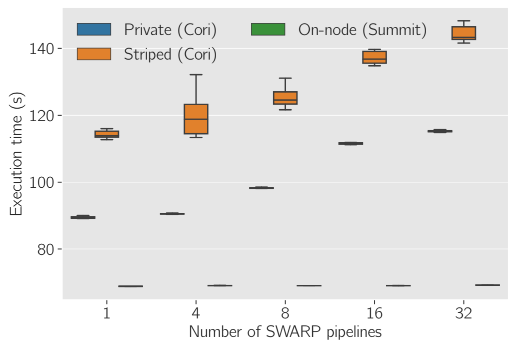


Figure 8. Resample variation when varying the number of pipelines. All files produced or read by the workflow tasks are into the burst buffer (On-node mode is the lowest).

On the difficulty of measuring I/O at scale on a shared machine. Obtaining I/O measurements is a challenging task, mainly due to the inherently concurrent aspect of I/O systems, the large number of processes, and continuous users interaction with I/O subsystems. Thus, the speed at which one accesses storage or memories is not deterministic and varies between executions. We investigated the potential difference between BB modes and implementations in terms of performance stability in the face of these types of interference (Figure 8). As expected, the on-node implementation outperforms the shared implementation by up to two orders of magnitude. The absence of network latency for the Summit BB architecture leads to more stable measurements. For the shared BB architecture, the private mode outperforms the striped mode (improvements of one order of magnitude), and further yields a more stable performance behavior – when using the striped mode execution performance can vary by $\sim$15%.

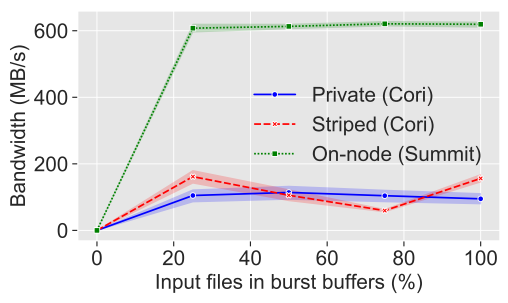


Figure 9. Average I/O performance (in term of bandwidth in MB/s) for Cori's shared implementation (private and striped), and Summit's on-node implementation.

Summary of findings. This experimental evaluation allows us to draw the following conclusions for the execution of the SWarp workflow on both Cori and Summit: (i) performance is highly dependent on access pattern and BB mode, in particular for the shared implementation; (ii) the number of cores plays an important role on performance but does not impact the relative performance trends obtained among the BB architectures; (iii) when increasing the number of pipelines the bandwidth becomes saturated although its usage is far below the peak; and (iv) in Cori, due to its shared aspect, the BB design is easy to leverage but hard to achieve peak performance (in part because performance is unstable when dealing with relatively small files, unlike when dealing with very large files [10]). On-node BB designs (e.g., Summit at ORNL) yield more stable performance (Figure 9), but they raise difficult resource allocation and data management challenges as sharing files across multiple on-node BB is not trivial.

IV. Modeling and Simulating Workflow Executions with Burst Buffers

In this section, we present a model for workflows execution on HPC systems equipped with burst buffers. To evaluate the accuracy of our model, we developed a simulator that models the studied experimental platform, which includes compute nodes, networking, and storage resources (PFS and BB). This simulator is built using the WRENCH simulation framework [33], [34], which can be used to build simulators of complex distributed systems that are accurate, can run scalably on a single computer, and can be implemented with minimal software development effort. To this end, WRENCH builds on the existing open-source SimGrid simulation framework [35], in which the high accuracy of its simulation models have been demonstrated via thorough invalidation and validation studies. The simulator code and experimental scenarios used in this section are all publicly available online [36].

A. Simulation instantiation

Our WRENCH simulator takes as input a description of a workflow and a description of an execution platform. The workflow description is a graph in which vertices are tasks and edges (i.e., task dependencies) are induces by input/output files of these tasks. Each task is annotated with a sequential compute time. Also, a parallel efficiency can be provided to estimate the compute time for a multi-core execution based on Amdahl's Law [37]. The description of the execution platform, an XML file, specifies all available compute resources (nodes, cores, RAM) and storage resources (disks), and the network resources (links, routers) that interconnect them. Based on this input, the simulator simulates the execution of the workflow and outputs a time-stamped event trace. The date of the last event, which corresponds to the last task completion, gives the overall makespan.

As stated above, our simulator expects as input, for each task, a measure of the sequential compute time (excluding I/O operations). However, from our experimental data (see Section III-D), we only know the observed execution time of a task on some number of cores, and the fraction of time that was spent in I/O operations. Therefore, we need a model for determining the task's purely computational sequential execution time based on the experimental data.

Therefore, we propose a simple, yet practical, model for determining the sequential compute time of a given task i knowing its observed fraction of time spent in I/O (λ_i^{io}) and the number of cores used (p). Let $T_i(p)$ be the *observed* execution time of task i using p identical cores, and let $T_i^c(p)$ be the raw compute time (without I/O) of task i (i.e., assuming an infinitely fast storage system). Here, $T_i(p)$ represents the execution time measured on the actual platform, and $T_i^c(1)$ is the required as input to the simulator, which is to be determined. We have the following relationship:

$$T_i^c(p) = (1 - \lambda_i^{io})T_i(p) \ , \tag{1}$$

where λ_i^{io} is the observed fraction of time this task spends doing I/O operations. We need to compute $T_i^c(1)$. As WRENCH uses a speedup model based on Amdahl's Law [37] to estimate the execution time of a parallel task, we use the same model[1]. Amdahl's Law states that the execution time of a task i running on p cores can be expressed as:

$$T_i^c(p) = \alpha_i T_i^c(1) + (1 - \alpha_i)\frac{T_i^c(1)}{p}, \tag{2}$$

where α_i is the fraction of the sequential execution time that cannot be parallelized. Thus, by equating the right-hand sides of Equations (1) and (2), we obtain:

$$T_i^c(1) = \frac{(1 - \lambda_i^{io})T_i(p)}{\alpha_i + (1 - \alpha_i)/p} \tag{3}$$

[1]On a side note, several other parallel execution models have been designed over the years, such as BSP [40] or LogP [41]. These models are more detailed than Amdahl's Law as they consider the network to model potential communication overhead occurring at scale, while Amdahl assumes that communications and computations behave similarly. However, these models require fine-grained knowledge about the execution platform in order to be instantiated, which in turn is impracticable. The approach chosen in this paper is to determine a first-order approximation of the raw sequential computation time to be fed to our simulator, which accurately models the I/O time and communication overheads.

Table I
INPUT PARAMETERS USED IN SIMULATION FOR EVALUATING THE ACCURACY OF OUR PROPOSED MODEL.

	Processor Speed	Burst Buffers Bandwidth		PFS Bandwidth	
		Network	Disk I/O	Network	Disk I/O
Cori [11]	36.80 GFlop/s/core	800 MB/s	950 MB/s [38]	1.0 GB/s	100 MB/s
Summit [12]	49.12 GFlop/s/core	6.5 GB/s	3.3 GB/s [39]	2.1 GB/s	100 MB/s

In this paper we make assumptions to simplify the above so that our model can be easily instantiated. A first important assumption is about parallel efficiency – we assume that all tasks follow a perfect speedup model (i.e., $s_i = 0$ for all i). Note that we had implicitly made the same assumption about I/O times (i.e., I/O time decreases linearly with the number of cores performing the I/O operations). Hence:

$$T_i^c(1) = p\,(1 - \lambda_i^{io})\,T_i(p). \tag{4}$$

These are quite strong assumptions that will definitely lead to losses in accuracy, as quantified later in this section. However, we aim to keep our approach simple, and platform- and application-agnostic. If the Amdahl's Law parameter (α_i) is known for each workflow task i, then the model in Equation (3) could also be used.

The value for the observed fraction of time spent doing I/O operations for each task (λ^{io}) is obtained from [24], in which an extensive characterization of SWarp on Cori I/O systems was conducted (using the PFS). The value of λ^{io} for Resample and Combine is 0.203 and 0.260, respectively. These values were obtained on Cori, but we also use them for Summit.

Finally, Table I shows the platform parameters for processor speeds, network bandwidths, and I/O bandwidths, passed to the simulator.

B. Accuracy

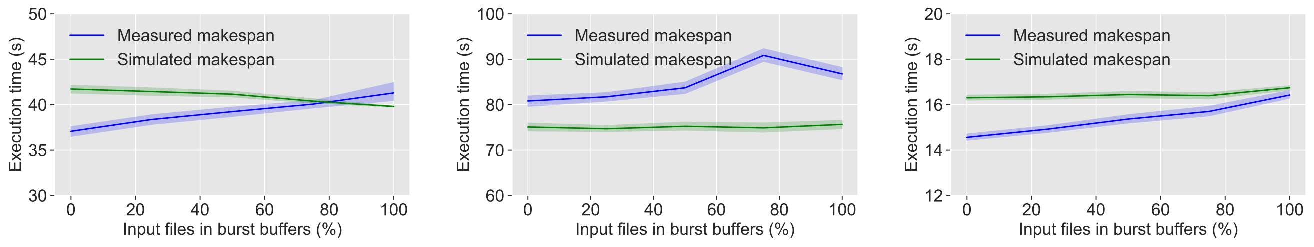
We assess the accuracy of our model and simulator by comparing the data gathered in Section III-D and the workflow makespan (turnaround time to compute all tasks in the workflow) produced by our simulator. We consider the exact same scenarios as that discussed in Section III-B,

Figure 10 shows the comparison between the predicted and measured makespan when varying the fraction of the input files that are staged into BBs. The corresponding experiment is shown on Figure 5. The proposed simulated model yields fairly accurate makespan – average error about 5.6% for the private mode, and 6.5% for the on-node implementation (Figures. 10(a) and 10(c)). Notice that the simulated makespan slightly overestimates the execution performance for these two scenarios – mostly due to the simplistic assumptions regarding I/O time. For the striped mode, we observed a larger error of about 12.8%, and an underestimation of the execution performance. This underestimation is mostly due to the fact our simulator does not account for the data fragmentation on the different nodes, thus the increased latency. Additionally, recall that the actual performance behavior of the striped mode for the Stage-In task yields an unexpected performance decrease when 75% of the files are staged into the BB (Figure 4). This behavior is not captured by our simulation, thus the increased error at 75%. The fact that the SWarp workflow reads/writes fairly small files (several MB) explain also the poor performance reached by the striped mode. We expect that with larger files (in the GB range), the striped mode would yield better performance. We note that Figure 10(a) is the only case in the results where the simulated makespan does not follow the same trend as the measured makespan. Although pinpointing the root cause of this discrepancy requires further investigation, we conjecture that the constant increase seen in the measure makespan is due to concurrent storage access. The files produced by the SWarp workflow might not be large enough to benefit from a private BB. By contrast, the simulator behaves as expected, the more the workflow uses burst buffers the faster it runs, hence leading to the opposite trends.

Figure 11 compares actual and simulated makespans when increasing the number of pipelines running on a single compute node. This configuration is particularly interesting for an I/O-focused study because by increasing the number of concurrent pipelines the likelihood that sharing interference occurs also increases. Although the overall average error is higher (11.8%, 11.6%, and 15.9% for private, striped, and on-node, respectively), the simulator framework yields predicted makespan trends similar to the actual execution. This result indicates that the competition for bandwidths among the concurrent pipelines is captured fairly well by our model and simulator. Also note that as the concurrency increases, the simulated makespans tend to becomes more accurate.

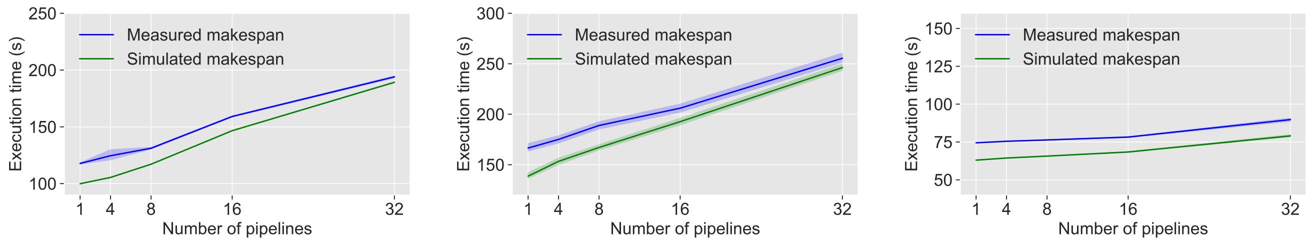
Although we could have tailored the model and simulator calibration parameters to specifically capture the observed behavior for the striped shared mode, we argue that this would have made the simulator platform- and configuration-dependent. More generally, we note that fine-grained performance and configuration details, required for tuning the calibration parameters, are often not available – e.g., in order to define the disk and network bandwidths for this evaluation (Table I) we have come across several documents that provided inconsistent information. Augmenting the simulator with additional parameters can only improve accuracy if it is possible to provide accurate value for these parameters, which often turns out to be difficult.

In summary, we argue that despite the simplifying assumptions in our model, our simulator yields satisfactory workflow makespan estimates while requiring minimal configuration parameters. More importantly, our evaluation experiments demonstrate that our proposed model and simulator can rea-



(a) Private (avg. error 5.64%, sd. 4.25%) (b) Striped (avg. error 12.84%, sd. 5.05%) (c) On-node (avg. error 6.55%, sd. 3.03%)

Figure 10. Real and simulated makespan when varying the number of files staged into BBs for one pipeline where each task runs on 32 cores. (Intermediate files produced by Resample and consumed by Combine are allocated into BBs.)



(a) Private (avg. error 11.82%, sd. 7.29%) (b) Striped (avg. error 11.59%, sd. 5.98%) (c) On-node (avg. error 15.93%, sd. 1.82%)

Figure 11. Real and simulated makespan when varying the number of pipelines. All files produced or read by the workflow are allocated into BBs and each task runs on a single core.

sonably capture relevant I/O behaviors trends.

C. Case-study for a large I/O-intensive workflow

In this section, we use our simulator to simulate the execution of a large data-intensive workflow on the Cori and the Summit architectures. To this end, we leverage execution traces of the 1000Genomes workflow [42] obtained from the WorkflowHub project [43], [44]. The 1000 genomes project [45] provides a reference for human variation, having reconstructed the genomes of 2,504 individuals across 26 different populations. The 1000Genomes workflow 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 (Figure 12). The workflow is composed of the following tasks: (i) *individuals:* downloads and parses data from the 1000 genomes project for each chromosome; (ii) *populations:* downloads and parses five super populations of individuals and a set of all individuals; (iii) *sifting:* computes the SIFT of all of the single nucleotide polymorphisms (SNPs) variants, as computed by the Variant Effect Predictor; (iv) *pair overlap mutations:* measures the overlap in mutations among pairs of individuals; and (v) *frequency overlap mutations:* calculates the frequency of overlapping mutations across subsamples of certain individuals. In this section we consider an instance of the 1000Genomes composed of 903 tasks, which processes 22 chromosomes. The total workflow data footprint of ∼67 GB.

Figure 13 shows the predicted makespan obtained when simulating the above 1000Genomes workflow instance on both

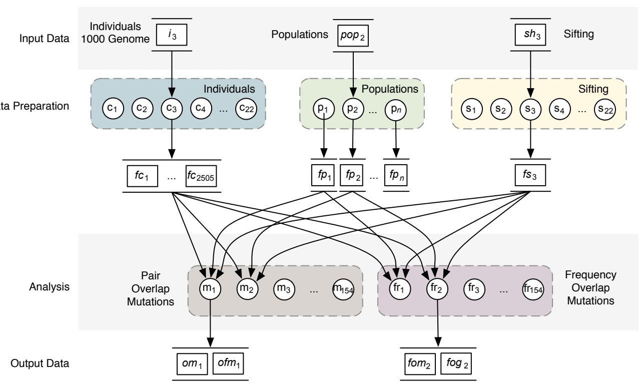


Figure 12. Overview of the 1000 genomes workflow

Cori and Summit. For this experiment, we vary the number of input files allocated in the BB. Note that these simulations were performed with the same configuration scenario and calibration parameters used to obtain the results shown in Figures 10 and 11. The performance of the workflow increases linearly as more files are allocated in the BB. As expected, Summit outperforms Cori mainly due to its larger BB bandwidth (see Table I). In addition, Cori reaches its performance plateau when about over 80% of the input data are allocated in the BB (bandwidth saturation). We conjecture that a striped BB allocation would improve the performance in this case by using more BB nodes and, therefore, alleviating the pressure on the bandwidth. On Summit, the performance

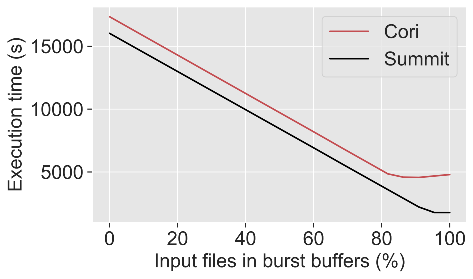


Figure 13. 1000Genomes workflow with 903 tasks (total input data is about 52 GB, i.e. 77% of the workflow data footprint).

plateau is reached later (when nearly all files are stored in the BB), which is expected due to its higher bandwidth.

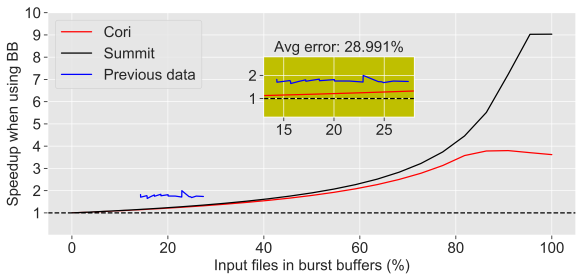


Figure 14. Speedup obtained when staging input data from the 1000Genomes workflow into BB nodes. Data from our previous study [10] are shown in blue for comparison purposes.

Figure 14 shows the same results as in Figure 13 but in terms of parallel speedup. This figure also shows, in blue, results obtained in [10], which conducted a thorough performance characterization study of the 1000Genomes workflow on Cori, using BBs, but only for a few different fractions of the buffer files stored in BBs. We show these results as interesting reference point, rather than attempting to use them for a thorough validation study, for the following reasons. First, the results in [10] were, for technical reasons, performed on a smaller 1000Genomes configuration (2 chromosomes) than that used in this section for our simulated executions (22 chromosomes). It turns out that, for 1000Genomes, different numbers of chromosomes modify the task-dependency structure and thus the I/O access patterns. Second, several aspects of the system (both hardware and software) have been upgraded in the time between when the experiments in [10] were conducted and the time when we performed the experiments in this work. Third, the load on the system is never the same between experiments, and definitely when experiments are many months apart.

Given all the above caveats, it is not surprising that the magnitude of the error is larger than that reported in the previous section. Nevertheless, at 29%, the error is not completely unreasonable, and one might expect that, were the results in [10] to be reproduced on Cori today with the 22-chromosome workflow configuration, lower error could be achieved. We plan to confirm this expectation experimentally in future work.

V. Conclusion

In this paper, we have explored the impact of next generation I/O systems on the performance of high-profile workflows applications. We have performed an extensive set of experiments running a representative real-world scientific workflow on two leadership-class HPC systems provided of two exemplary BBs I/O systems (Cori at NERSC, and Summit at ORNL). We have conducted a comprehensive analysis of this workflow and explored the parameter-space to underline several challenges that should be addressed by workflows and applications that target those systems. We have then proposed a model for I/O-intensive workflows that accounts for the use of BBs on HPC systems. We used the gathered data to calibrate and validate an open-source simulator. We have thoroughly exposed the differences between the two access shared modes available on Cori, and compared their performance to Summit's on-node implementation; and discussed their advantages and limitations. In particular, we have showed that the striped mode can, depending on the application's I/O patterns, lead to poor performance and to potential substantial sharing interference. On the other hand, Summit's on-node approach yields far better and more stable performance. We have also evaluated the accuracy of our proposed model via simulation on complex scenarios, including an evaluation of our approach by contrasting simulated behavior to actual executions of a large-scale data-intensive workflow previously executed on a BB system. A natural future direction is to leverage our simulator to explore the heuristic-space of data placements strategies to optimize workflows executions, and to quantify the resulting benefits.

Acknowledgments

This work is funded by DOE contract number #DESC0012636; and partly funded by NSF contracts #1664162, #1741040, #1923539, and #1923621. 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. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. We would also like to thank C. Daley and L. Ramakrishnan from NERSC for their help with the SWarp workflow.

References

[1] A. Barker and J. Van Hemert, "Scientific workflow: a survey and research directions," in *PPAM*, 2007.

[2] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A survey of data-intensive scientific workflow management," *Journal of Grid Computing*, vol. 13, no. 4, pp. 457–493, 2015.

[3] J. S. Vetter, R. Brightwell, M. Gokhale, P. McCormick, R. Ross, J. Shalf, K. Antypas, D. Donofrio, T. Humble, C. Schuman *et al.*, "Extreme Heterogeneity 2018 - Productive Computational Science in the Era of Extreme Heterogeneity: Report for DOE ASCR Workshop on Extreme Heterogeneity," Lawrence Berkeley National Lab.(LBNL), Tech. Rep., 2018.

[4] J. Y. Choi, C.-S. Chang, J. Dominski, S. Klasky, G. Merlo, E. Suchyta, M. Ainsworth, B. Allen, F. Cappello, M. Churchill *et al.*, "Coupling exascale multiphysics applications: Methods and lessons learned," in *2018 IEEE 14th International Conference on e-Science (e-Science)*. IEEE, 2018, pp. 442–452.

[5] J. Bent, G. Grider, B. Kettering, A. Manzanares, M. McClelland, A. Torres, and A. Torrez, "Storage challenges at los alamos national lab," in *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2012, pp. 1–5.

[6] K. Sato, N. Maruyama, K. Mohror, A. Moody, T. Gamblin, B. R. de Supinski, and S. Matsuoka, "Design and modeling of a non-blocking checkpointing system," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–10.

[7] A. Moody, G. Bronevetsky, K. Mohror, and B. R. De Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2010, pp. 1–11.

[8] T. Wang, K. Mohror, A. Moody, K. Sato, and W. Yu, "An ephemeral burst-buffer file system for scientific applications," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 807–818.

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

[10] R. Ferreira da Silva, S. Callaghan, T. M. A. Do, G. Papadimitriou, and E. Deelman, "Measuring the impact of burst buffers on data-intensive scientific workflows," *Future Generation Computer Systems*, vol. 101, pp. 208–220, 2019.

[11] NERSC, "Lawrence Berkeley National Laboratory's Supercomputer Cori," https://www.nersc.gov/users/computational-systems/cori.

[12] J. Wells, B. Bland, J. Nichols, J. Hack, F. Foertter, G. Hagen, T. Maier, M. Ashfaq, B. Messer, and S. Parete-Koon, "Announcing supercomputer summit," 6 2016.

[13] H. Jia-Wei and H.-T. Kung, "I/o complexity: The red-blue pebble game," in *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, 1981, pp. 326–333.

[14] A. Benoit, S. Perarnau, L. Pottier, and Y. Robert, "A performance model to execute workflows on high-bandwidth-memory architectures," in *Proceedings of the 47th International Conference on Parallel Processing*, 2018, pp. 1–10.

[15] J. Herrmann, L. Marchal, and Y. Robert, "Memory-aware list scheduling for hybrid platforms," in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. IEEE, 2014, pp. 689–698.

[16] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *012 ieee 28th symposium on mass storage systems and technologies (msst)*. IEEE, 2012, pp. 1–11.

[17] B. Nicolae, A. Moody, E. Gonsiorowski, K. Mohror, and F. Cappello, "Veloc: Towards high performance adaptive asynchronous checkpointing at large scale," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS*. IEEE, 2019, pp. 911–920.

[18] K. Sato, K. Mohror, A. Moody, T. Gamblin, B. R. De Supinski, N. Maruyama, and S. Matsuoka, "A user-level infiniband-based file system and checkpoint strategy for burst buffers," in *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2014, pp. 21–30.

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

[20] M.-A. Vef, N. Moti, T. Süß, M. Tacke, T. Tocci, R. Nou, A. Miranda, T. Cortes, and A. Brinkmann, "Gekkofs—a temporary burst buffer file system for hpc applications," *Journal of Computer Science and Technology*, vol. 35, no. 1, pp. 72–91, 2020.

[21] J. Peng, S. Divanji, I. Raicu, and M. Lang, "Simulating the burst buffer storage architecture on an ibm bluegene/q supercomputer," *SC16, Supercomputing16*, 2016.

[22] L. Cao, B. W. Settlemyer, and J. Bent, "To share or not to share: comparing burst buffer architectures," in *Proceedings of the 25th High Performance Computing Symposium*, 2017, pp. 1–10.

[23] R. Ferreira da Silva, S. Callaghan, and 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.

[24] C. S. Daley, D. Ghoshal, G. K. Lockwood, S. Dosanjh, L. Ramakrishnan, and N. J. Wright, "Performance characterization of scientific workflows for the optimal use of burst buffers," *Future Generation Computer Systems*, 2017.

[25] D. Ghoshal and L. Ramakrishnan, "MaDaTS: Managing data on tiered storage for scientific workflows," in *26th International Symposium on High-Performance Parallel and Distributed Computing*, 2017.

[26] J. Cope, N. Liu, S. Lang, P. Carns, C. Carothers, and R. Ross, "Codes: Enabling co-design of multilayer exascale storage architectures," in *Proceedings of the Workshop on Emerging Supercomputing Technologies*, vol. 2011. ACM, 2011.

[27] N. Liu, C. Carothers, J. Cope, P. Carns, R. Ross, A. Crume, and C. Maltzahn, "Modeling a leadership-scale storage system," in *International Conference on Parallel Processing and Applied Mathematics*. Springer, 2011, pp. 10–19.

[28] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan, "Scalable i/o forwarding framework for high-performance computing systems," in *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, 2009, pp. 1–10.

[29] M. Mubarak, P. Carns, J. Jenkins, J. K. Li, N. Jain, S. Snyder, R. Ross, C. D. Carothers, A. Bhatele, and K.-L. Ma, "Quantifying i/o and communication traffic interference on dragonfly networks equipped with burst buffers," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2017, pp. 204–215.

[30] G. Aupy, O. Beaumont, and L. Eyraud-Dubois, "What size should your buffers to disks be?" in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 660–669.

[31] H. Khetawat, C. Zimmer, F. Mueller, S. Atchley, S. S. Vazhkudai, and M. Mubarak, "Evaluating burst buffer placement in hpc systems," in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2019, pp. 1–11.

[32] T. Wang, "Exploring novel burst buffer management on extreme-scale hpc systems," Ph.D. dissertation, The Florida State University, 2017.

[33] H. Casanova, S. Pandey, J. Oeth, R. Tanaka, F. Suter, and R. Ferreira da Silva, "Wrench: A framework for simulating workflow management systems," in *13th Workshop on Workflows in Support of Large-Scale Science (WORKS'18)*, 2018, pp. 74–85.

[34] H. Casanova, R. Ferreira da Silva, R. Tanaka, S. Pandey, G. Jethwani, W. Koch, S. Albrecht, J. Oeth, and F. Suter, "Developing Accurate and Scalable Simulators of Production Workflow Management Systems with WRENCH," *Future Generation Computer Systems*, 2020.

[35] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, 2014.

[36] "Workflow I/O BB simulator," https://github.com/lpottier/workflow-io-bb.

[37] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, 1967, pp. 483–485.

[38] C. V. Canada, J. M. Patchett, R. K. Braithwaite, J. P. Ahrens, and M. Ruiz Varela, "Beyond defensive io: Leveraging the burst buffer for in-transit visualization workflows," https://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-15-28099, 2015.

[39] "Samsung PM1725a NVMe SSD," https://www.samsung.com/semiconductor/global.semi.static/Brochure_Samsung_PM1725a_NVMe_SSD_1805.pdf, 2018.

[40] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[41] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. Von Eicken, "Logp: Towards a realistic model of parallel computation," in *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, 1993, pp. 1–12.

[42] R. Ferreira da Silva, R. Filgueira, E. Deelman, E. Pairo-Castineira, I. M. Overton, and M. Atkinson, "Using simple pid-inspired controllers for online resilient resource management of distributed scientific workflows," *Future Generation Computer Systems*, vol. 95, pp. 615–628, 2019.

[43] "WorkflowHub: Community Framework for Enabling Scientific Workflow Research and Education," https://workflowhub.org, 2020.

[44] R. Ferreira da Silva, W. Chen, G. Juve, K. Vahi, and E. Deelman, "Community resources for enabling and evaluating research in distributed scientific workflows," in *10th IEEE International Conference on e-Science*, ser. eScience'14, 2014, pp. 177–184.

[45] The 1000 Genomes Project Consortium *et al.*, "A global reference for human genetic variation," *Nature*, 09 2015.