Evaluating Burst Buffer Placement in HPC Systems

Harsh Khetawat[1], Christopher Zimmer[2], Frank Mueller[1], Scott Atchley[2], Sudharshan S. Vazhkudai[2], Misbah Mubarak[3]
[1]North Carolina State University, [2]Oak Ridge National Laboratory, [3]Argonne National Laboratory
{hkhetaw, fmuelle}@ncsu.edu, {zimmercj, atchleyes,vazhkudaiss}@ornl.gov, mmubarak@anl.gov

***Abstract*—Burst buffers (BBs) are increasingly exploited in contemporary supercomputers to bridge the performance gap between compute and storage systems. The design of BBs, particularly the placement of these devices and the underlying network topology, impacts both performance and cost. As the cost of other components such as memory and accelerators is increasing, it is becoming more important that HPC centers provision BBs tailored to their workloads.**

This work contributes a provisioning system to provide accurate, multi-tenant simulations that model realistic application and storage workloads from HPC systems. The framework aids HPC centers in modeling their workloads against multiple network and BB configurations rapidly. In experiments with our framework, we provide a comparison of representative Oak Ridge Leadership Computing Facility (OLCF) I/O workloads against multiple BB designs. We analyze the impact of these designs on latency, I/O phase lengths, contention for network and storage devices, and choice of network topology.

***Index Terms*—HPC, burst buffers, simulation, I/O**

I. Introduction

With the increasing scale of computation and data in High Performance Computing (HPC), existing storage systems are becoming a bottleneck for large-scale scientific and data-intensive applications [1]. The ratio of I/O bandwidth to bytes of memory capacity on Titan (the 27 petaflops (PF) system at Oak Ridge Leadership Computing Facility, OLCF, currently at No. 12 in the Top500 list with a 1 TB/s filesystem) is 0.0016 and the ratio on Summit (the 200 PF system at OLCF, currently No. 1 in the Top500 list with a 2.5 TB/s filesystem) is 0.0001. Data generation rates are increasing faster than traditional parallel file system (PFS) ingestion capabilities. To alleviate this bottleneck, the traditional PFS is being augmented with a tier of intermediate, high-bandwidth flash-based storage devices called burst buffers (BBs). These BBs sit between compute nodes and the parallel file system (PFS), and are designed to absorb the periodic I/O bursts of HPC applications.

BBs allow applications to checkpoint their state more quickly and frequently to persistent storage and data to be staged for input and output, enabling the application to resume computation rather than wait for I/O. The functional advantages and disadvantages of these architectures have been studied in previous work [2]. In large-scale HPC centers, BBs are a multi-million dollar resource that impact the center's productivity, the I/O performance of the application workload and the scientific progress of the users. The efficacy of BBs and their I/O performance depend on optimal provisioning and architectural details.

Extant BB provisioning solutions are based on either simple rules of thumb or simplistic scenarios (single application I/O behavior in isolation), and are not representative of the complexity involved in the design space. In this paper, we argue that an optimal provisioning of BBs for a large-scale HPC center should carefully consider and reconcile a variety of factors. Figure 1 illustrates the dimensions system designers and practitioners should reconcile during BB provisioning. For example, careful consideration must be given to aspects such as the locality of BBs, the underlying network topology, the application workload's I/O characteristics, and the job scheduling mix of the respective facility. Failure to do so will result in sub-optimal I/O solutions and slowdown of scientific workflows.

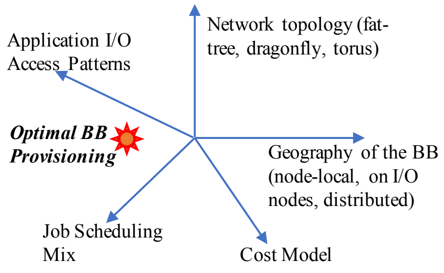

Network topology (fat-tree, dragonfly, torus)
Application I/O Access Patterns
Optimal BB Provisioning
Geography of the BB (node-local, on I/O nodes, distributed)
Job Scheduling Mix
Cost Model

Fig. 1. BB Provisioning Dimensions

To understand the design space further, let us consider the geographic locality of BBs within the HPC system. Various hierarchies exist for the placement of BBs in HPC systems: (a) BBs co-located with compute nodes, e.g., used in Summit, the next-generation OLCF system [3]; (b) BB nodes located alongside I/O nodes, e.g., used in Cori, an HPC system at NERSC [4]; (c) global BBs. In terms of network topology, modern HPC systems use a range of network topologies, e.g., fat-tree [5], dragonfly [6], torus, etc. These network topologies have different blocking characteristics, diameters, bisection bandwidths, and costs. The performance of BBs is most closely affected by the network topology, and then the application workload's I/O access patterns (N-N, N-1, etc.), the scheduling mix of the jobs and the interplay of their traffic. All of the above factors play a vital role in the eventual BB experience that user applications perceive.

In this work, we present a more rigorous approach to BB provisioning that carefully reconciles the aforementioned vectors. We have created a provisioning framework using Parallel Discrete Event Simulation (PDES), which provides a tool to HPC system designers allowing them to rapidly model

978-1-7281-4734-5/19/$31.00 ©2019 IEEE

their workloads against different BB architectures, placement strategies, and network configurations. System designers can then use the framework to make provisioning decisions. Using our simulations and models, we can evaluate various architectures to determine the ideal choices based on the application workload, performance requirements, and cost constraints.

The primary contributions of this work are as follows:

• the development of a complete framework that system designers and practitioners can use to input BB locality, network topology, I/O workload patterns, job scheduling mix and cost to study "what-if" scenarios for BB provisioning.

• the development of a variety of network and BB models (e.g., node-local) for a large-scale HPC system, and then simulating them via the CODES [7] [8] suite (and extending it) to assess their performance for varying workloads;

• the ability to replay workloads composed of *multiple* applications with customizable node allocation policies to accurately model an HPC center;

• the development of a novel capability to *strongly and weakly scale traces* of the Darshan I/O logs with the simulator to project future workloads for larger I/O sizes than any I/O traces collected on today's platform;

• the development of novel features within Darshan to replay I/O traces with semantics for *barrier-based blocking I/O collectives* to improve replay accuracy.

Besides assessing system configurations for procurement (as in this paper), the framework can further facilitate the development of parallel file systems, data staging schemes, traffic shaping algorithms and resiliency techniques by performing sensitivity studies to parametric variations.

II. Background

In this section we will discuss tiering in modern HPC storage systems and HPC network topologies. We will also discuss the functional advantages and disadvantages that different burst buffer placement techniques have and how they impact the network interconnect.

HPC Storage Tiers: Bursty I/O traffic from applications has been reported to create a bottleneck at the level of shared disk-based parallel file systems [1]. In order to absorb these spikes in I/O traffic, the addition of a fast tier of SSD-based storage, called BBs, has been both proposed and implemented [9] [10]. BBs are closer to the compute nodes than the PFS and offer the applications significantly higher bandwidth, albeit at a much lower capacity. Applications that read/write a large amount of data can now keep the compute nodes busy by utilizing the high-bandwidth SSD-based storage for checkpointing, reading input data, writing preliminary results, and their final output to persistent storage. BBs also allow applications to perform significantly faster in-situ analysis and jobs in a workflow to have a high-bandwidth, low-latency scratchpad.

Several leadership-scale computing systems have deployed BBs. Theta at the Argonne National Laboratory, Summit at the Oak Ridge National Laboratory, and Sierra at the Lawrence Livermore National Laboratory have node local BBs. On the other hand, Trinity at the Los Alamos National Laboratory, Conrad at the Zuse Institute Berlin, and Cori at the Lawrence Berkeley National Laboratory have distributed BBs.

Network Topologies: The performance of HPC applications is often limited by communication rather than computation making the network interconnect a key determinant of system performance. A host of network topologies exist, including tori, trees and meshes plus recent ones like dragonfly, slimfly [11], and HyperX [12]. Each one of these topologies offer different characteristics such as diameter, bisection bandwidth, direct/indirect network, and cost (see Table I).

TABLE I
Network Topologies

Network	Bisection	Diameter	Network Type	Cost
3-level Fat-Tree	Full	4	Indirect	High
3D Torus	Low	High	Direct	Low
Dragonfly	High	3	Direct	Medium
Slimfly	High	2	Direct	Medium

The choice of network topology and allocation of jobs to nodes has a significant impact on system performance. Furthermore, placement of BB nodes can impact network contention in these interconnect topologies. Therefore, it becomes important to study the impact of application communication, BB I/O traffic and shared parallel file system traffic on them.

Application Workloads

Leadership computing facilities run a variety of application workloads, from large jobs that occupy a significant fraction of the system to several small to medium sized jobs each of which occupy a smaller fraction of the system. Workloads can also vary by their duration of execution from a few hours to a few days. These factors have an impact on the frequency and size of I/O traffic that these applications have.

Applications also have different checkpointing requirements, input/output characteristics, and I/O stages that affect the overall performance of the application workload. The use of traces from leadership class machines allows us to replicate application behavior at a high resolution.

III. Related Work

Prior work has focused on the exploration and implementation of BB architectures in HPC systems. Kimpe et al. [13] describe the design of a container abstraction to manage in-system storage devices and to transfer data in the storage hierarchy. Herbein et al. [14] use I/O aware batch scheduling to reduce contention with novel scheduling techniques. This reduces contention on the parallel file system, resulting in reduced job variability. BurstMem [15] provides storage and communication strategies for BB systems. It shows that if handled efficiently BBs can significantly speed up application I/O performance. TRIO [16] is another framework that coordinates flushing from BBs to the parallel file system in order to maximize storage bandwidth by reducing the contention between storage servers. In contrast, we focus on the impact of BB placement on application performance.

Bhimji et al. [10] explore the use of the Cori BB system at the National Energy Research Scientific Computing Center (NERSC) of the Lawrence Berkeley National Laboratory. They discuss the performance gains achieved using BBs compared to only a shared Lustre parallel file system. Liu et al. [9] have used parallel discrete event simulation to evaluate BBs in leadership-class storage systems. They analyze common burst patterns in applications and use simulation to analyze the I/O performance of applications. While Liu et al. delve into application performance in the presence of a BB system, our work focuses on creating a reproducible framework for analyzing application and BB performance for different BB placement strategies and network configurations.

More recently, Mubarak et al. [17] have used simulations to show the effects of interference of network and I/O traffic in dragonfly network topologies equipped with BBs. They use different routing strategies with realistic workload sizes to demonstrate that balancing I/O and network traffic requires a careful selection of routing policies, and job and data placement. Harms et al. [2] describe the use cases of BBs and how different BB architectures are more suited for certain functionality than others. Cao et al. [18] compare the performance of local and shared BB systems. Their results show that shared BB organizations can result in higher I/O throughput than local BBs. Instead of static checkpoint sizes, we use application traces collected from real executions to more accurately simulate application I/O behavior.

IV. Overview

As part of this work, we create a framework using CODES to rigorously examine placement of storage resources in modern HPC systems. The network models allow HPC centers to simulate current and proposed HPC topologies. The burst buffer models in our framework also facilitate a what-if comparison of various burst buffer architectures against which HPC centers can simulate their workloads. HPC centers can use Darshan traces collected from previous application executions to project application I/O behavior with significant resolution to future systems. Our framework can also be used to augment the Darshan traces with MPI synchronization primitives, to increase the number of I/O phases, and to scale the traces via extrapolation, both strongly and weakly. These capabilities allow for projections to future workload sizes while preserving application behavior. Our framework can aid burst buffer provisioning by allowing HPC centers to study various network and storage architectures with multi-job workloads and node allocation policies specific to the center.

V. CODES Simulation Suite

Parallel Discrete Event Simulation: In Discrete Event Simulation (DES) [19], the system is modeled as a sequence of discrete events. Each event changes the state of the system. Since these events occur at particular instances in time and trigger state changes, the system state is assumed to be constant between state changes. Parallel Discrete Event Simulation (PDES) [20] exploits the parallelism to significantly speed-up simulation performance, while also allowing us to scale the simulation to larger sizes. On the other hand, PDES is hard to implement because some events might affect others, and therefore require sequencing constraints. Without sequencing constraints, causality errors can occur.

Two mechanisms manage sequencing constraints — conservative, and optimistic. For conservative PDES, causality errors are prevented from occurring by issuing an event only when it is safe. Events are not processed until all events that might affect it have completed. Alternatively, optimistic PDES allows causality errors to occur but has mechanisms to detect such errors and roll back to a correct state.

Rensselaer's Optimistic Simulation System (ROSS): ROSS [21] is a DES that uses Time Warp [22] for synchronization. Each component in the system is modeled as a logical process (LP), which communicates by exchanging timestamped event messages. It has support for both sequential and parallel (conservative and optimistic) simulations.

The Time Warp mechanism synchronizes computation by detecting events that occurred out of timestamp order, rolling-back these events, and finally re-executing them. In order to improve performance, ROSS uses a technique called Reverse Computation [23] [24]. In Reverse Computation, instead of saving and recovering state in case of causality errors, roll-back is done by reverse executing code. This allows us to scale the simulation to highly parallel machines and saves memory as states between events need not be preserved.

CODES: CODES builds on ROSS in order to enable highly parallel simulations of exascale network and storage architectures in HPC environments. CODES abstracts the network models as components to create packet-level simulations of the most popular HPC network topologies. It supports dragonfly, slimfly, torus and fat-tree topologies. It includes support for packetization of messages and provides an API to simulate both MPI and RPC style communication. It also has support for storage models including a local storage and a CODES store model, which can be used to simulate BBs.

Once the HPC system has been modeled using the storage and network components, CODES can simulate a range of I/O and network workloads. CODES has support for synthetic workloads, checkpoint workloads as well as replaying network traces such as SST DUMPI and I/O traces from Darshan. For our experiments, we use Darshan traces from real- world HPC applications to evaluate the performance of BBs. Finally CODES allows for the collection of several metrics related to the simulation, which we use for our evaluation.

Darshan: Darshan [25] is an I/O characterization tool developed by researchers at Argonne National Laboratory that allows application developers and HPC system administrators to capture an accurate picture of the I/O behavior of an application. It consists of two parts, a runtime, which is a lightweight library used to instrument the application at execution time, and the Darshan utility, which is a collection of tools used to analyze Darshan traces.

The Darshan runtime is lightweight enough to be used on several current generation HPC systems in order to instrument I/O behavior and has been deployed at the Argonne Leadership Computing Facility (ALCF), the National Energy Research Scientific Computing Center (NERSC), and the Oak Ridge

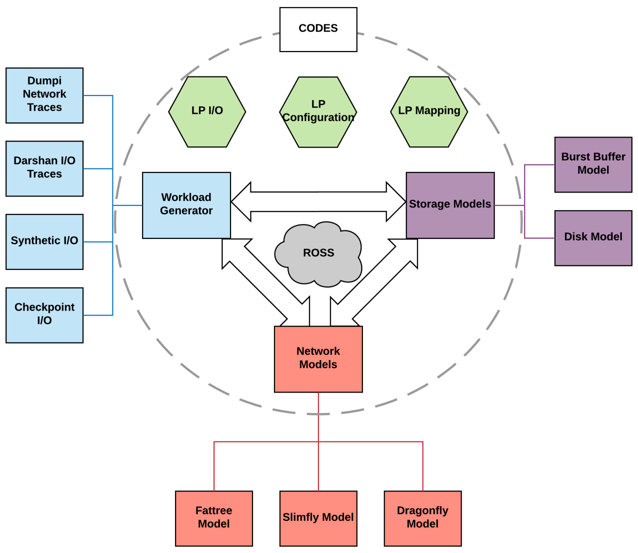


Fig. 2. The CODES simulation suite

Leadership Computing Facility (OLCF), among others. We use the traces collected by these systems to resemble the I/O behavior of scientific applications in our simulation. We also use the Darshan utility to scale the Darshan traces collected by these HPC centers for future, larger scale HPC systems.

VI. Workload and Simulation Design

A. BB Placement

The placement of BBs in an HPC system has a significant impact not only on the I/O performance of the application but also on the performance of the interconnect. In order to evaluate the performance of the placement models, we simulate each model under varied configurations. Their performance is then evaluated with different workloads and network topologies (described later). Each model has its own functional advantages and disadvantages, which we also discuss. Figure 3 shows the different BB architectures we are evaluating in a fat-tree network configuration.

1) Node-Local BBs: A storage device is placed within the compute node allowing applications to have exclusive access to the storage resource while enabling I/O performance to scale linearly with the number of nodes in the system. Further, I/O operations do not result in network traffic as it goes through the local bus. One drawback of using this model is that the BBs are tightly coupled with the compute nodes, which creates a single failure domain. It also makes it difficult to support applications that utilize shared files.

To simulate the node-local BB model, we integrate the CODES storage server into the client. Since reading/writing from/to the local buffer does not result in network traffic, one can model the BB in this manner. The I/O is modeled to be synchronous, and application execution resumes after I/O completes. Bandwidth, latency and seek times of storage devices are configurable via the CODES configuration.

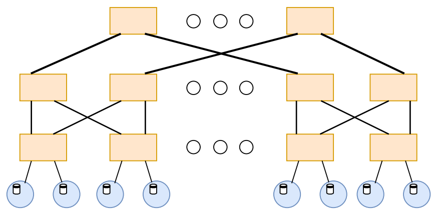
2) Grouped BBs: BB servers are placed in the group along with the compute nodes in order to exploit local network links, which are typically non-blocking. This model allows applications to write to shared files and supports easy stage-in and stage-out. It is also much more straightforward to share data between nodes. Here, the bursty I/O traffic could negatively impact the PFS as buffers share network connections. This configuration incurs an additional server, networking, and cabling costs. We simulate this model by placing the CODES storage server model in each group along with the client LPs (logical processes). The CODES storage server is built using the local storage model that simulates a disk along with networking and has support for threading, which simulates multiplexed transfers. Memory size, storage size, threads, read/write bandwidth, latency, and seek times are configurable in the CODES configuration file. Read/writes from compute nodes result in network traffic to I/O nodes.

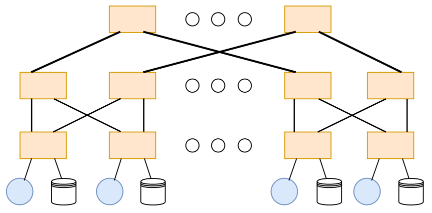
3) Global BBs: BB servers are placed on nodes separate from both the compute nodes and the I/O nodes. This allows the compute nodes to use shared files, and the BBs might still be usable even if the parallel file system goes down. The bursty I/O here results in additional traffic on the global network, which could cause degraded performance. It also increases the cost of the system due to additional servers, network interface cards (NICs), and network switches and cables. The simulation of global BBs is done using the same CODES storage server placement as in the last model, except that the BB servers are placed in a separate group with a configurable number of burst buffer nodes. The I/O in this model causes network traffic to be generated from the compute nodes to the BBs. A significant difference from the previous model is that flushing data from the BBs to the parallel file system would result in network traffic from the BB nodes to the I/O nodes. Parameters such as memory, storage, bandwidth, etc. are configurable in the CODES configuration file.

4) Locality and Striping: For the node-local model, the MPI ranks on a node conduct I/O with their local BBs without accessing the network. In the grouped model, ranks of the application perform I/O with the BB device located in the same group as well as with the BB devices in the adjacent groups based on the striping configuration. Similarly, for the global BB model, the application performs I/O with a respective BB node from the set of global BBs as well as a certain number of adjacent nodes as specified in the striping configuration. In our experiments, the I/O in the node-local model is not striped across multiple BB devices, i.e., the only cause of contention is the SSD device on the BB node. For grouped and global BB models, we stripe the I/O across 4 BB nodes with a stripe width of 128KB. This configuration closely mirrors that of common HPC storage hierarchies including Data Direct Network's (DDN) Infinite Memory Engine (IME).

B. Workloads

Gyrokinetic Toroidal Code, GTC [26], is a highly scalable scientific application that simulates billions of plasma particles inside a reactor. S3D [27] is a direct numerical simulation (DNS) code used in computational fluid dynamics, which solves the Navier-Stokes equations. LAMMPS [28] stands for Large-scale Atomic/Molecular Massively Parallel Simulator


(a) Node-Local (BB in leaf nodes)


(b) Grouped (BB in adjacent nodes)

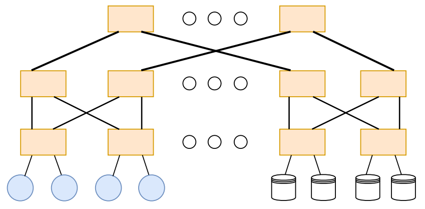

(c) Global (BB is dense node subset)

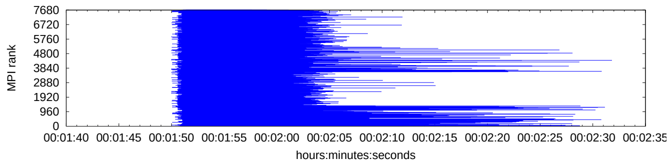
Fig. 3. BB architectures on a fat-tree network.

and is used to model particles at the mesoscale or continuum levels. HACC or Hardware/Hybrid Accelerated Cosmology Code [29] is used to conduct high resolution simulations of cosmological structure for modern-day galactic surveys. IOR is a benchmarking application developed by LLNL to test the performance of parallel file systems. We use it to simulate adversarial traffic in the system allowing us to study the impact of artificially high I/O and network traffic on applications. In order to evaluate the effects of different workloads on the I/O performance of the system, we use HPC applications widely used in leadership computing facilities. We use Darshan traces collected from runs of the application to replay the expected I/O behavior on our simulated systems. These representative applications perform blocking I/O. We also created tools to weakly scale the Darshan logs to occupy a specific fraction of the system. We discuss the scaling approach in a later section.

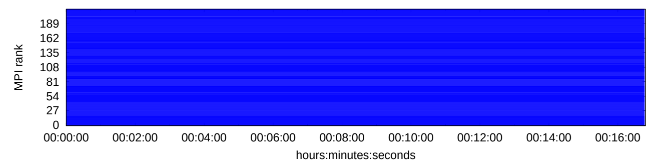
TABLE II
APPLICATION TRACES

Application	MPI Ranks	Avg. I/O / Rank (MB)
GTC	7680	669.41
LAMMPS	21600	19.78
S3D	6000	154.49
HACC	8192	386.39
IOR	768	1024

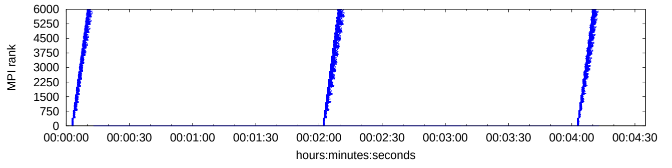
Table II lists the applications used in our workloads and their I/O properties, and Table III lists the combinations of these applications we use as representative workloads for our simulation experiments. Figure 4 shows the I/O patterns of our representative applications as derived from the Darshan traces. GTC has a single I/O phase towards the end of the execution of the application with all ranks (except rank 0) writing very similar amounts of data to persistent storage. Each rank in LAMMPS opens several files for writing small amounts of data into each file. S3D has 3 I/O phases for each rank. Except rank 0, each of the ranks writes to an independent file in each of the 3 I/O phases. The write operations are also staggered with increasing rank numbers. For HACC, all the ranks have 2 write I/O phases to the same file. While each of the ranks open their respective files at the same time, the operations exhibit a scattered pattern with each rank experiencing a delay before write operations commence. This significantly reduces the overlap between writes from different ranks. Finally, IOR as a benchmark has a single I/O phase with each rank writing 1 GB to independent files from the very beginning of application execution. The nodes are modeled to be Summit-style (200 petaflop CPU/GPU system at OLCF) fat nodes with 6 GPUs, i.e, we allocate 6 MPI ranks to each compute node in our simulation. Our choice for application



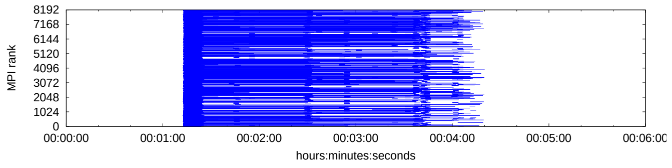
(a) GTC



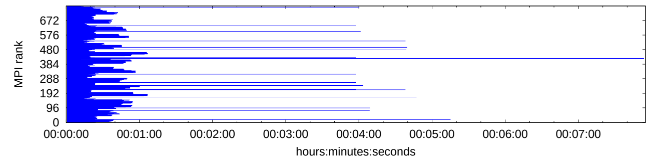
(b) LAMMPS



(c) S3D



(d) HACC



(e) IOR

Fig. 4. File write patterns for different applications from Darshan traces.

combinations were restricted by the number of nodes in the system we are simulating and memory limitations on our compute nodes.

TABLE III
HPC WORKLOADS

Workload	Total MPI Ranks	Compute Nodes	%age of Total Nodes
GTC	7680	1280	23-27
GTC, IOR	8448	1408	25-30
GTC, S3D	13680	2280	40-47
GTC, S3D, HACC	21872	3646	64-75
HACC	8192	1366	25-29
HACC, IOR	8960	1494	27-32
LAMMPS	21600	3600	63-74
LAMMPS, IOR	22368	3728	65-77
S3D	6000	1000	17-21
S3D, IOR	6768	1128	20-23
S3D, LAMMPS	27600	4600	80-95

1) Augmenting Darshan Traces: More powerful machines allow application developers to increase the size of their datasets, create higher fidelity simulations, and perform analysis at finer granularity. To more effectively represent application I/O behavior in next-generation machines, we develop tools that allow us to scale Darshan application traces and add synchronization primitives to them. First, our tool allows us to scale the number of ranks in an application trace. Following the weak scaling paradigm, it uses the I/O behavior of the other ranks to scale-up the number of ranks within a job. Second, we can also scale the size of the I/O per rank in the job for strong scaling. Third, we added the ability to repeat the number of iterations that the Darshan trace is replayed for with a certain interval. And finally, we have the ability to collate several consecutive small writes in the trace into bigger ones. We can, therefore, represent future application I/O sizes while being consistent with application I/O behavior. We can also add global and sub-communicator barriers to the traces to more accurately represent I/O behavior of the application.

2) Job Placement: Applications in leadership-class systems are rarely allocated to a completely contiguous set of nodes. The HPC batch scheduling algorithm often results in fragmentation with several small chunks of contiguous nodes that are available. This exacerbates inter-job and intra-job interference (see Yang et al. [30]) for MPI communication. This can also have severe effects on I/O performance with interference resulting in large performance variations for non node-local BB configurations and the PFS. We use job allocation logs from the Titan supercomputer at OLCF to guide the allocation of applications to nodes in our workload. This allocation strategy lets us evaluate the impact of application I/O patterns not only on interference in the network but also in terms of interference between SSD devices of BBs.

C. System Configuration

In this section, we describe the systems we simulate using the CODES simulation suite. We broadly categorize the systems based on their network topology. For each of the network topologies, we create configurations for the different BB architectures subject to evaluation.

1) Fat-Tree Network: We use the fat-tree model in CODES to configure a fully non-blocking fat-tree network resembling Summit. Our configuration is a 3 level fat-tree with 270 edge switches (radix of 36) and up to 4860 terminal nodes. Like Summit, our configuration has a network bandwidth of 25 GB/s per node. For the node-local BBs, each of the terminals has compute nodes with a local SSD. The grouped configuration has one BB server per edge switch, and the global BB configuration has 15 edge switches with only BB servers for a total of 270 BB servers. For effective comparison we have both the configurations, grouped and global, with the same number of BB nodes. Furthermore, all the 3 configurations have nearly identical aggregate bandwidth (within 1% of each other.)

2) Dragonfly Network: We use the dragonfly model in CODES to simulate a Cray XC30 system along the lines of the Edison supercomputer at NERSC. It uses an Aries-style interconnect for a total of 5,760 terminal nodes with a bandwidth of 8 GB/s per node. The local and global channels are configured with a bandwidth of 5.25 GB/s and 1.5 GB/s, respectively. We use adaptive routing which has been shown to perform better than minimal routing for adversarial traffic patterns [6]. Here, the node-local BBs are also situated on each compute node. The grouped configuration has a BB for every 2 blades in the network, where each blade consists of 4 terminal nodes connected by an Aries SOC. The global BB configuration consists of 180 blades with BB servers.

3) A 2:1 Tapered Fat-Tree Network: We also use the fat-tree model in CODES to configure a 2:1 tapered fat-tree with twice the number of links for terminal nodes as the links going to the upper-level switches. This reduces the cost by reducing the number of switches and cables required for the same number of terminal nodes. Tapering of a fat-tree might negatively impact performance of the network as it is no longer non-blocking. To facilitate a comparison with a full non-blocking fat-tree network, the 2:1 tapered fat-tree retains the other configuration parameters from our fat-tree configuration.

4) BB Nodes: Node-local BBs are simulated as a single SSD device per compute node with write and read bandwidths of 1,400 MB/s and 2,300 MB/s, respectively, for both dragonfly and fat-tree networks. The grouped and global BB nodes consist of 18 SSDs each for the fat-tree configuration and 6 SSDs each for the dragonfly configuration. The aggregate write and read bandwidth of BB nodes is 25,000 MB/s and 41,000 MB/s in the fat-tree configuration and 8,400 MB/s and 13,700 MB/s in the dragonfly configuration, respectively, for both the grouped and global BB models.

VII. Evaluation

We evaluate the performance of different workloads subject to a range of factors. We run 100 simulation experiments on an HPC cluster using as input application traces collected from runs on the Titan supercomputer at OLCF [31].

First, we look at how the performance of single job workloads are impacted by the choice of BB architectures. We compare the architectures across the different network topologies based on total time spent on I/O, network hops and BB performance. We then study the impact of adversarial jobs on our representative workloads. This involves a workload co-scheduled with an IOR job, which periodically writes 1 GB files to BBs, enabling us to gauge performance in a worst-case scenario. Finally, we study the impact of co-scheduling a combination of representative jobs on performance parameters.

A. Validation of Node-Local BB Model

To validate our node-local BB model, we run the IOR benchmark on Summit-style [3] nodes in a fat-tree network with node-local BBs. We run the benchmark with 96 ranks over 16 nodes, and compare the results with those obtained from simulation. Each rank writes 1GB of data to independent files on their local BBs. We also set the parameters of our simulated BBs to match the configuration of the actual devices.

We set the read/write bandwidth to 3,200 MB/s and 2,100 MB/s respectively with negligible seek overhead.

Table IV shows the comparison between the simulated and actual run of the IOR benchmark on 16 nodes. We can see that the write time and bandwidth obtained from the simulation deviates from the actual results by less than 3%. This validates our model to simulate node-local BBs with considerable accuracy.

TABLE IV
NODE-LOCAL BB VALIDATION

Parameter	Actual	Simulated
Size/Rank (GB)	1	1
Aggregate Size (GB)	96	96
Write Time (s)	3.01	2.92
Aggregate Bandwidth (MB/s)	32600.56	33616.48

B. Validation of Distributed BB Models

We validate our distributed BB models (which combines grouped and global from Fig. 3 for now) by running the IOR benchmark on the Konrad [32] TDS (Test Development System) for 6 - 192 ranks over 64 compute nodes. The Konrad TDS has an Aries Dragonfly network topology and distributed BBs. We use their BB configuration within CODES for a write bandwidth of 1,900 MB/s and 15μs of write overhead, based on a single node with 6 ranks writing to one SSD of the BB, in order to validate against the performance of the actual machine. Figure 5 shows the comparison between the actual and simulated runs of the IOR benchmark. The results from the simulation deviate from the actual runs by less than 11%. This validates our CODES model to simulate global and grouped distributed BBs accurately.

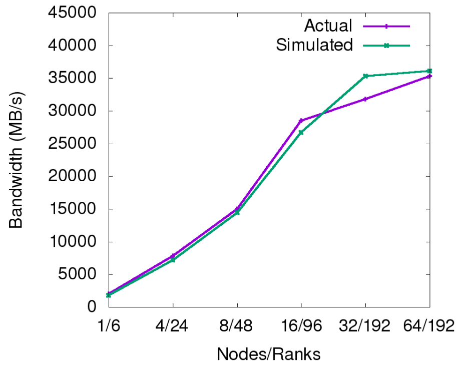


Fig. 5. Distributed BB Validation

1) Cost Modeling: In a cost-benefit analysis, we consider DOE models of workloads for supercomputing centers. Capability computing centers cater to a workload that requires 20% or more of the resources in a supercomputer to achieve the desired science. These centers tune scheduling to prioritize large jobs to access the machine. These workloads often use file-per-process outputs to mitigate performance degradation due to parallel file system locking at scale. Capacity computing center resources are targeted toward smaller average job sizes dominated by single-shared-file outputs. Smaller jobs are less impacted by file system overheads and single-shared-files do not require post-processing like file-per-process.

We make the following assumptions for cost modeling that are based on Summit [3]. We assume that distributed BB file-per-process performance and single-shared file performance are equivalent, and there is no locking overhead. We assume that there is no opportunity to perform single-shared file operations on node-local systems. Using Summit's storage performance numbers at 2.5 TB/s for PFS and 9.7 TB/s for the node-local BB, a 35% memory checkpoint takes approximately 5 minutes to the parallel file system and 2 minutes to the burst buffer based on related work [33]. Over the course of a year, such workloads in a capacity center result in 100% acceleration of the I/O portions of the workload by 2.5X. In contrast, the capability center accelerates the I/O portions by 1.8X as only 70% of the I/O workload can be accelerated (on the BB side).

Table V compares Capital Expenditure (CAPEX) and the Operational Expenditure (OPEX) of the node local and distributed models using current hardware. The numbers presented are a simplification of actual OPEX values and do not account for voltage scaling during idle phases. These approximations also omit the OPEX maintenance costs that are necessary for running such systems. The numbers demonstrate that the distributed model's CAPEX is 2.4X higher than the node-local model. The energy-based OPEX is also higher, consuming over double the power due to CPU, network, and infrastructure overheads. Compared to node local BBs, a capacity center using a distributed BB would more than double the CAPEX and OPEX. A capability center (same CAPEX+OPEX) would only benefit from 70% of the I/O being accelerated, still making distributed BBs less appealing.

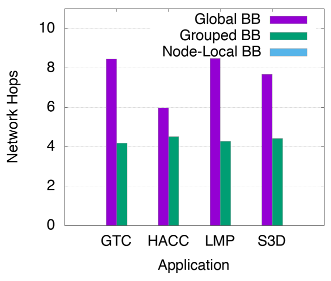
TABLE V
DISTRIBUTED VS. NODE-LOCAL CAPITAL AND OPERATIONAL EXPENDITURE COMPARISON

Node Local CAPEX	Total Cost	$4,608,000
4608	x8 NVMe Devices	$1000
Distributed CAPEX	Total Cost	$11,415,600
4832	x4 NVMe Devices	$800
302	AMD EPYC Servers	$25,000
Node Local OPEX	5 Yr Cost	$1,000,000
NVMe	4608	25W
	Total	115KW
Distributed OPEX	5 Yr Cost	$2,600,000
NVMe	4832	25W
Server	604	300W
	Total	302KW

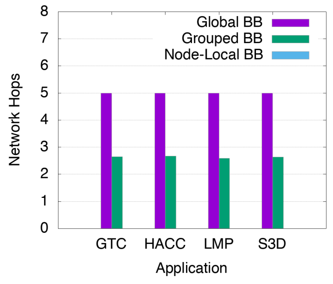
C. Effect of BB Placement on Performance

We simulated the 4 representative jobs (GTC, HACC, LAMMPS and S3D) running independently on the system for each BB architecture and network topology. The node allocations are not contiguous but guided by allocation logs from the Titan supercomputer, which are typically not contiguous in node locations. This provides a realistic comparison when we introduce adversarial jobs as well as co-schedule other applications as part of the workload.

Figure 6 shows the average number of hops incurred by I/O traffic for each of the given applications across the different BB architectures. Since node-local traffic does not have to traverse the network, the number of hops for node-local BB model is 0. For the fat-tree network, all traffic to the global BBs is routed across the entire diameter of the network. For the grouped BBs traffic is routed to a combination of the BB servers in the local group as well as adjacent groups because of the striping policy.



(a) Dragonfly



(b) Fat-Tree

Fig. 6. Avg. number of network hops incurred by application's I/O traffic

Observation 1: For bursty traffic in a dragonfly network with global BBs we observe high non-minimal routing, with I/O traffic incurring significantly higher hops.

For the dragonfly network we see significant differences between applications for the global BB architectures. GTC exhibits highly bursty traffic, performing significant I/O in a relatively short duration of time. This causes a higher fraction of the packets to be routed non-minimally compared to HACC, which exhibits staggered rather than bursty behavior. The grouped architecture, on the other hand, does not concentrate traffic in any one part of the network and, therefore, does not suffer from excessive non-minimal routing. Dragonfly networks attempt minimal routing first, followed by non-minimal routing. Therefore, high non-minimal routing indicates congestion in the network.

Observation 2: Grouped BB in a dragonfly topology perform their I/O phases significantly faster than the global configurations for bursty applications.

Figure 7 shows the I/O phase of each rank of the application on the y-axis (from highest rank on the top to lowest rank on the bottom) over time (x-axis) for a single I/O phase. As expected, the performance of the node-local BBs is independent of the choice of network topology across all applications. Ranks in the grouped architecture spend consistently less time in their I/O phase than with the global BB architecture for the three network topologies. The difference is most pronounced for the dragonfly topology with bursty applications. This is due to the lower global bandwidth on the dragonfly topology compared to the fat-tree with full bisection bandwidth. In case of HACC, not all of the ranks attempt write operations at the same time. This limits the network contention in the part of the network where the storage resources are concentrated.

Observation 3: Global, grouped and node-local BB configurations have comparable performance for fat-tree networks except for applications with staggered I/O like S3D.

In the case of fat-tree topology, all applications except S3D spend comparable amounts of time in their I/O phases for the three BB architectures. For S3D, the grouped and global configurations outperform the node-local BB configuration for both the full fat-tree and the 2:1 tapered fat-tree networks. Since only a fixed number of ranks in S3D (400 in this case) perform I/O at any given time, there is more available bandwidth in the case of global and grouped BB configurations, while the application is bottle-necked by the single SSD in the node-local BBs. The tapering imposes a performance penalty on both grouped and global BB configurations, but the network contention between the edge and aggregation layers results in the global configuration paying a significantly higher penalty compared to the grouped BB configuration.

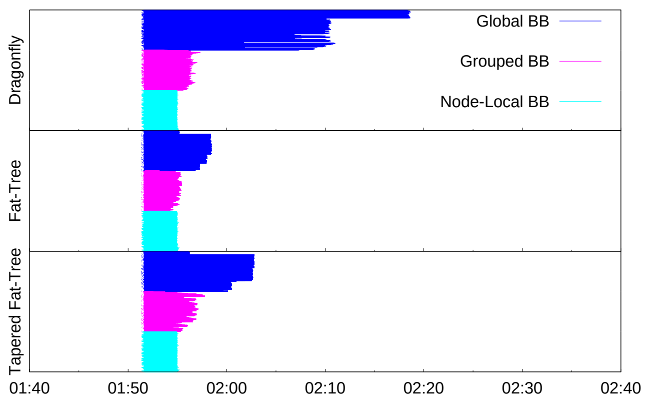
Observation 4: Node-local BBs outperform the other configurations for bursty applications. In case of applications with a scattered I/O phase like HACC, the node-local configuration results in each rank spending more time on I/O.

We can also see from Figure 7 that node-local outperforms grouped and global BB architectures for bursty applications like GTC and LAMMPS. This is because (a) the SSD is local to the compute node and I/O does not incur slowdown due to network congestion, and (b) there is much less contention for the SSD device compared to grouped and global BBs. For HACC, results show that each rank in the node-local configuration spends significantly more time on I/O operations compared to grouped or global BBs but the resulting I/O phase lengths are almost identical irrespective of BB placement. This is because while all ranks in the application open their respective output files at the same time, they commence write operations only after a delay of anywhere between 1 to 40 seconds. Due to this scattering of write operations by ranks, network and storage contention is reduced drastically. Since the node-local BBs have lower bandwidth compared to BB servers in the grouped or global architectures, each rank spends more time on I/O. In this case, HACC has a schedule that perfectly matches bandwidth restrictions of the grouped and global models, as a mismatch would have caused performance to be adversely affected, which is not the case. We conjecture that an I/O runtime that manages scheduling of I/O operations can potentially benefit grouped and global BB architectures.

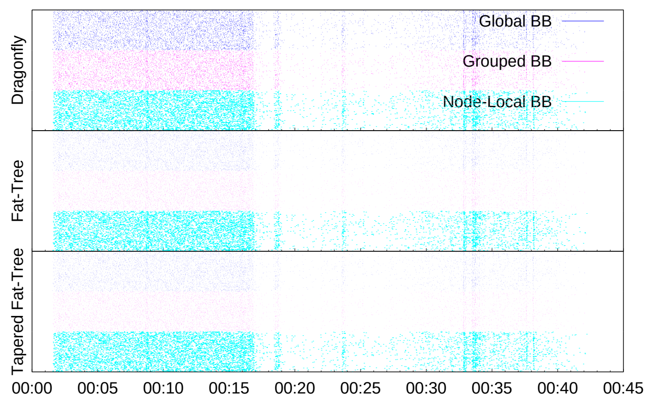
Our framework can also model different storage devices like Phase Change Memory (PCM) and 3D XPoint. To assess diverse BB devices, we experimented with a cost (overhead) of $45\mu s$ per operation for BBs (graphs omitted due to space). In this case, node-local BBs significantly outperform the other configurations for bursty applications and staggered applications like S3D. Furthermore, global and grouped BBs in a fat-tree configuration had similar performance across applications for storage devices with seek and operational overhead.

D. Effect of Co-Scheduled Jobs on Performance

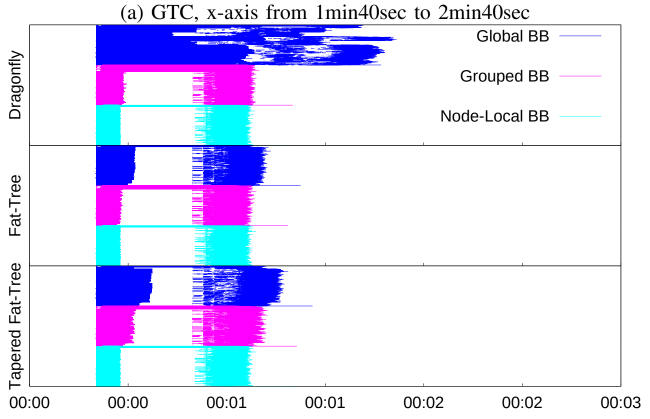
We also simulate the performance of the representative applications when scheduled alongside other applications plus an adversarial one. Here, the simulated storage device has additional read and write overheads of $45\mu s$ per operation. This allows us to experiment with emerging storage technologies that have different characteristics than current generation SSDs (without measurable write overheads). We use traces of IOR, an I/O benchmark generating large volumes of data at specified intervals to simulate the adversarial traffic. The node allocation strategy causes co-scheduled jobs to be assigned to certain nodes adjacent to the nodes of the application we are studying. Since the co-scheduled jobs have no effect on the



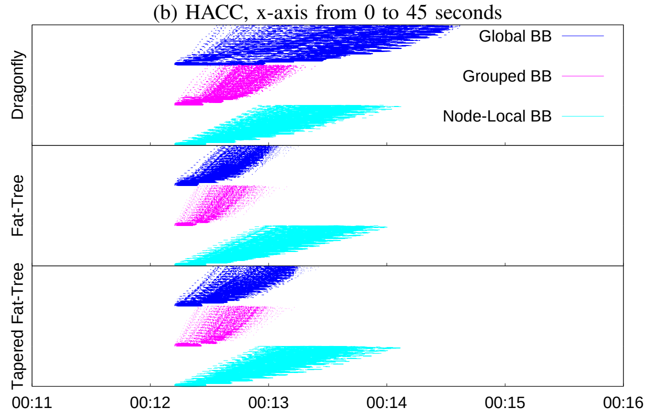
(a) GTC, x-axis from 1min40sec to 2min40sec



(b) HACC, x-axis from 0 to 45 seconds



(c) LAMMPS, x-axis from 0 to 3 seconds



(d) S3D, x-axis from 11 to 26 seconds

Fig. 7. I/O phases of each application across network topologies and BB architectures, y-axis from highest (top) to lowest (bottom) rank

performance of the node-local BBs, we show results only for global and group-local BBs.

Figure 8 shows the effect of scheduling multiple jobs on application workloads. The different configurations are specified by network type and BB architecture. DF, FT, and TFT refer to the dragonfly, fat-tree, and tapered fat-tree networks, respectively. The figures show the total duration of the I/O phase (in seconds) on the y axis for each application, rather than on a per-rank basis.

Observation 5: Applications with I/O phases that do not overlap significantly have little to no effect on the application's I/O phase length even for bursty applications.

For GTC, the I/O phase of the application has little overlap with the traffic from co-scheduled jobs. This minimizes interference between jobs. Even though some ranks complete their I/O phases sooner and others later, the average time spent by each rank is not significantly affected. Even though individual ranks spend more time in I/O for certain ranks of the GTC application, the overall I/O phase remains similar.

Observation 6: The I/O phase lengths of non-bursty/scattered applications are not significantly affected by co-scheduled jobs, though there is an increase in the average time spent in I/O by each rank.

For HACC, the scattered I/O pattern results in ranks adjacent to the nodes running other jobs spending more time on their respective I/O phases, especially for fat-tree and tapered fat-tree topologies, which is not visible from the aggregate plots. The overall I/O phase of the application remains unaffected due to the highly scattered nature of I/O operations even as the average time spent in the I/O phase increases. Since the I/O behavior of HACC is not bursty, it is almost unaffected by the I/O behavior of other applications running on the system.

Observation 7: Bursty applications in fat-tree and tapered fat-tree networks are significantly affected by co-scheduled adversarial jobs due to contention for both storage and network resources.

LAMMPS, on the other hand, experiences significant interference from the adversarial job. The I/O phase of ranks adjacent to nodes allocated to the adversarial job increases significantly, causing the application to experience slowdown. While the interference is experienced across BB architectures and network topologies, the effect is most pronounced for the fat-tree networks. In case of the tapered fat-tree network, the limited network bandwidth from the edge to the aggregate layers results in similarly degraded performance for both the global and the grouped BB configurations. The bursty nature of LAMMPS coupled with its I/O phase coinciding with that of the IOR job results in contention for both storage and network resources. Although experiencing significant slowdown due to IOR, LAMMPS is not at all affected by the a co-scheduled S3D job as their I/O phases do not coincide.

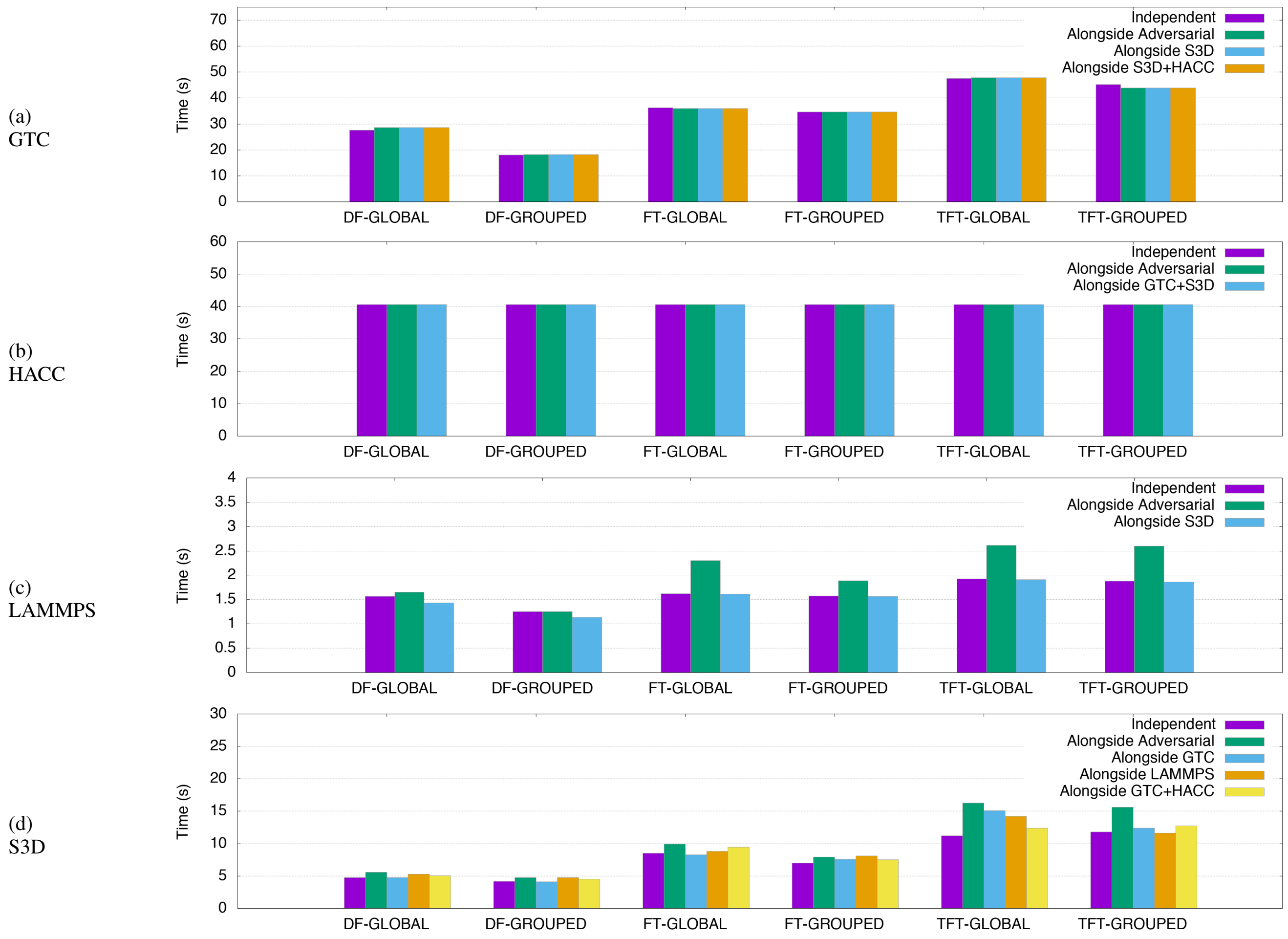


Fig. 8. I/O phases of each application with adversarial background workload across network topologies and BB architectures.

Observation 8: The I/O performance of applications with custom patterns, like S3D, is dependent on the node allocation policy employed by HPC centers.

S3D experiences significantly degraded performance due to an IOR job being scheduled along side S3D, especially for the fat-tree networks. Just as for LAMMPS, network contention in the tapered fat-tree network causes similarly degraded performance for both the global and grouped BB configurations. We can also see from the figure that the tapered fat tree configuration imposes the highest penalty compared to the dragonfly network, where the performance degradation is the least. This indicates high network contention rather than contention at the storage resources. Furthermore, a co-scheduled GTC job has little to no effect on the I/O performance of S3D in dragonfly and fat-tree networks, but significantly affects performance for both global and grouped BB configurations in the tapered fat-tree network. Due to the nature of I/O in S3D, if the co-scheduled job is allocated nodes adjacent to the higher S3D ranks, the long tails of those ranks would result in increased I/O phases of the entire application.

Finally, we also perform experiments with SSD devices with 0 seek and operational overhead. Our preliminary results show that HACC is not significantly affected by the storage device model because of the scattered I/O pattern. Similarly, LAMMPS's I/O phase length decreases, reducing the overlap with the adversarial jobs, and is therefore not less affected by the adversarial job as in Fig. 8, which includes $25\mu s$ and $20\mu s$ seek and operational overheads.

VIII. Conclusion

We have developed a simulation framework for the provisioning of burst buffers in supercomputers to provide accurate, multi-tenant evaluations of realistic application and storage workloads. This allows us to compare multiple network and BB configurations as a means to select a best configuration for procurement of an HPC system based on the target workloads to run. Our experiments indicate difference depending on application characteristics, such as I/O burstiness, BB placement, overlap of I/O phases, background workloads, and network topology with an intricate interplay, all of which aid in ultimately deciding on a network topology and BB placement.

Acknowledgment

This work was funded in part by NSF grants 1217748 and 1525609, a subcontract from LLNL and by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The work used the resources from the OLCF.

References

[1] B. Xie, J. Chase, D. Dillow, O. Drokin, S. Klasky, S. Oral, and N. Podhorszki, "Characterizing output bottlenecks in a supercomputer," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. IEEE, 2012, pp. 1–11.

[2] K. Harms, H. S. Oral, S. Atchley, and S. S. Vazhkudai, "Impact of burst buffer architectures on application portability," Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States). Oak Ridge Leadership Computing Facility (OLCF), Tech. Rep., 2016.

[3] O. R. N. Laboratory, "Summit," https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/.

[4] N. E. R. S. C. Center, "Cori," http://www.nersc.gov/users/computational-systems/cori/.

[5] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE transactions on Computers*, vol. 100, no. 10, pp. 892–901, 1985.

[6] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*. IEEE, 2008, pp. 77–88.

[7] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns, "Enabling parallel simulation of large-scale hpc network systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 87–100, 2017.

[8] 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, 2011.

[9] 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 *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*. IEEE, 2012, pp. 1–11.

[10] 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*, 2016.

[11] M. Besta and T. Hoefler, "Slim fly: A cost effective low-diameter network topology," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 348–359.

[12] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "Hyperx: topology, routing, and packaging of efficient large-scale networks," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, 2009, p. 41.

[13] D. Kimpe, K. Mohror, A. Moody, B. Van Essen, M. Gokhale, R. Ross, and B. R. De Supinski, "Integrated in-system storage architecture for high performance computing," in *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*. ACM, 2012, p. 4.

[14] S. Herbein, D. H. Ahn, D. Lipari, T. R. Scogland, M. Stearman, M. Grondona, J. Garlick, B. Springmeyer, and 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.

[15] T. Wang, S. Oral, Y. Wang, B. Settlemyer, S. Atchley, and 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.

[16] T. Wang, S. Oral, M. Pritchard, B. Wang, and W. Yu, "Trio: burst buffer based i/o orchestration," in *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*. IEEE, 2015, pp. 194–203.

[17] 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 *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*. IEEE, 2017, pp. 204–215.

[18] 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*. Society for Computer Simulation International, 2017, p. 4.

[19] J. O. Henriksen, R. M. O'Keefe, C. D. Pegden, R. G. Sargent, B. W. Unger, and D. W. Jones, "Implementations of time (panel)," in *Proceedings of the 18th conference on Winter simulation*. ACM, 1986, pp. 409–416.

[20] R. M. Fujimoto, "Parallel discrete event simulation," *Communications of the ACM*, vol. 33, no. 10, pp. 30–53, 1990.

[21] C. D. Carothers, D. Bauer, and S. Pearce, "Ross: A high-performance, low-memory, modular time warp system," *Journal of Parallel and Distributed Computing*, vol. 62, no. 11, pp. 1648–1669, 2002.

[22] D. R. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 7, no. 3, pp. 404–425, 1985.

[23] C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto, "Efficient optimistic parallel simulations using reverse computation," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 9, no. 3, pp. 224–253, 1999.

[24] M. D. Peters and C. D. Carothers, "Parallel distributed simulation and modeling methods: an algorithm for fully-reversible optimistic parallel simulation," in *Proceedings of the 35th conference on Winter simulation: driving innovation*. Winter Simulation Conference, 2003, pp. 864–871.

[25] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 characterization of petascale i/o workloads," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–10.

[26] Z. Lin, T. S. Hahm, W. Lee, W. M. Tang, and R. B. White, "Turbulent transport reduction by zonal flows: Massively parallel simulations," *Science*, vol. 281, no. 5384, pp. 1835–1837, 1998.

[27] D. Lignell, C. S. Yoo, J. Chen, R. Sankaran, M. Fahey, E. Hawkes, T. Lu, and C. Law, "S3d: Petascale combustion science, performance, and optimization collaborators," 06 2018.

[28] S. Plimpton, P. Crozier, and A. Thompson, "Lammps-large-scale atomic/molecular massively parallel simulator," *Sandia National Laboratories*, vol. 18, pp. 43–43, 2007.

[29] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka *et al.*, "Hacc: Simulating sky surveys on state-of-the-art supercomputing architectures," *New Astronomy*, vol. 42, pp. 49–65, 2016.

[30] X. Yang, J. Jenkins, M. Mubarak, X. Wang, R. B. Ross, and Z. Lan, "Study of intra-and interjob interference on torus networks," in *Parallel and Distributed Systems (ICPADS), 2016 IEEE 22nd International Conference on*. IEEE, 2016, pp. 239–246.

[31] O. R. N. Laboratory, "Titan," https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan/.

[32] Z. I. Berlin, "Konrad," http://www.zib.de/features/supercomputing-zib.

[33] S. S. Vazhkudai, B. R. de Supinski, A. S. Bland, A. Geist, J. Sexton, J. Kahle, C. J. Zimmer, S. Atchley, S. Oral, D. E. Maxwell, V. G. V. Larrea, A. Bertsch, R. Goldstone, W. Joubert, C. Chambreau, D. Appelhans, R. Blackmore, B. Casses, G. Chochia, G. Davison, M. A. Ezell, T. Gooding, E. Gonsiorowski, L. Grinberg, B. Hanson, B. Hartner, I. Karlin, M. L. Leininger, D. Leverman, C. Marroquin, A. Moody, M. Ohmacht, R. Pankajakshan, F. Pizzano, J. H. Rogers, B. Rosenburg, D. Schmidt, M. Shankar, F. Wang, P. Watson, B. Walkup, L. D. Weems, and J. Yin, "The design, deployment, and evaluation of the coral pre-exascale systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 52:1–52:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=3291656.3291726