Quantifying I/O and Communication Traffic Interference on Dragonfly Networks Equipped with Burst Buffers

Misbah Mubarak*, Philip Carns*, Jonathan Jenkins*, Jianping Kelvin Li†, Nikhil Jain§, Shane Snyder*, Robert Ross*, Christopher D. Carothers‡, Abhinav Bhatele§, and Kwan-Liu Ma†

*Argonne National Laboratory, USA {mubarak, carns, jenkins, ssnyder, rross}@mcs.anl.gov
†University of California, Davis, USA {kelli, klma}@ucdavis.edu
‡Rensselaer Polytechnic Institute, USA chrisc@cs.rpi.edu
§Lawrence Livermore National Laboratory, USA {jain6, bhatele}@llnl.gov

***Abstract*—HPC systems have shifted to burst buffer storage and high-radix interconnect topologies in order to meet the challenges of large-scale, data-intensive scientific computing. Both of these technologies have been studied in detail independently, but the interaction between them is not well understood. I/O traffic and communication traffic from concurrently scheduled applications may interfere with each other in unexpected ways, and this behavior may vary considerably depending on resource allocation, scheduling, and routing policies.**

In this work, we analyze I/O and network traffic interference on burst-buffer-equipped dragonfly-based systems using the high-resolution packet-level simulation provided by the CODES storage and interconnect simulation framework. The analysis is performed using realistic I/O workload sizes, a variety of resource allocation and network routing strategies employed in production environments, and a dragonfly network configuration modeled after current vendor options. We analyze the impact of interference on both I/O and communication traffic.

We observe that although average network packet latency is stable across a wide variety of configurations, the maximum network packet latency in the presence of concurrent I/O traffic is highly sensitive to subtle policy changes. Our simulations reveal a worst-case single packet latency of 4,700 times the average latency for sub-optimal configurations. While a topology-aware mapping of compute nodes to burst buffer storage nodes can minimize the variation in maximum packet latency, it can slow down the I/O traffic by creating contention on the burst buffer nodes. Overall, balancing I/O and network performance requires careful selection of routing, data placement, and job placement policies.

***Keywords*-burst buffer, dragonfly networks, discrete-event simulation, checkpoint, I/O and communication traffic**

I. Introduction

Recently, extreme-scale computing has also become data-intensive computing, as the size of datasets consumed and produced by experiments and simulations running on today's high-performance computing (HPC) systems has grown. As a result, next-generation supercomputers are moving to alternative storage designs to the venerable globally shared parallel file system model. As part of this shift, the concept of "burst buffers" is being adopted in leadership-class machines. Burst buffers form a new storage tier in HPC systems that serve as an intermediate, high-bandwidth store between computational resources and the primary parallel file system and are aimed to absorb periodic high-intensity I/O phases ("bursts") from applications, alleviating the need of parallel file systems to scale to short-lived peak I/O rates. For example, both the Cori and Trinity systems (being deployed at Lawrence Berkeley and Los Alamos National Laboratories, respectively) have deployed burst buffers using solid state drives (SSDs) [1], [2].

One of the primary motivations for burst buffer deployment is to improve application reliability by enabling faster application checkpointing/restarting [3], a fundamental resiliency strategy to retain computational progress upon component failures. Understanding the behavioral implications of burst buffers is crucial for designing performant checkpoint/restart approaches. However, recent shifts in the storage and interconnect design of emerging HPC systems raise numerous questions. For example, leadership systems are also moving to high-radix, low-diameter network topologies using nondeterministic routing for load balancing, such as the dragonfly topology [4]. While the research community has performed targeted studies of these designs [5], [6], [7], [8], [9], [10], much remains to be learned about the interaction between subsystem designs (storage placement on the network topology). More important, little is known about the interference of communication traffic with the I/O traffic being directed to the burst buffer nodes. To the best of our knowledge, this is the first study that is aimed at quantifying I/O and network traffic interference with different configurations of dragonfly networks equipped with burst buffer nodes.

Evaluating resource allocation, job placement, and routing policies on real hardware requires dedicated access to such systems for an extended period of time, with the ability to orchestrate complex scenarios. Because regularly obtaining this type of access is not feasible, the research community has relied on predictive design techniques such as analytical modeling and simulation; the cited studies all rely on such techniques. While these techniques can assist in the design and deployment of exascale systems, there are very few performance prediction tools that can accurately

reflect complex HPC applications and architectures with high fidelity at a large scale while executing in a reasonable time. In this paper, we build our models using the CODES interconnect and storage simulation suite that provides the ability to efficiently execute detailed, large-scale discrete-event simulations.

The primary contributions of this paper are the following:

- We have extended the CODES simulation suite to support a general-purpose burst buffer storage model that provides concurrent, pipelined RDMA read and write operations.
- We have extended the CODES packet-level dragonfly network model to support vendor designs, such as the ones offered in Cray Cori and Theta systems [1], [11]. A detailed validation report of the dragonfly network model on the Theta Cray XC systems can be found at [12].
- Using the extended simulation suite, we present a study of communication and I/O interference on a large-scale simulated HPC environment that executes multiple workloads, comprising background network traffic being generated at up to 12.5 GiB/s message rate and an aggregate of 1 TB being generated for I/O checkpoint traffic.
- We examine the following policy/design points, with the goal of quantifying performance characteristics of our case study workloads : (i) allocation of burst buffer nodes to application nodes, (ii) application placement on the dragonfly topology, and (iii) routing strategies on the dragonfly.

While exploring the plausible permutations of routing, job placement and data placement policies for the I/O traffic, our key findings are the following: (i) certain configurations get efficient I/O performance but lead to significant perturbation to background communication traffic, in some cases delaying individual network packets by a factor of 4700 or more; (ii) configurations that cause least perturbation to the communication traffic can substantially slow down the I/O traffic; and (iii) a careful choice of routing, data, and job placement policies can achieve minimal perturbation to communication traffic while efficiently utilizing the burst buffers for I/O.

II. Background & Motivation

This section provides an overview of the recent shifts in HPC interconnect and storage hierarchy, which serves as a motivation for the interference study conducted in this paper. It also discusses the existing research work carried out in the areas of burst buffers management and HPC networks.

A. HPC Storage Hierarchy

The HPC memory/storage hierarchy has recently undergone changes to support the spikes in I/O bandwidth generated by modern science applications [13], [14]. With the recent addition of SSD-based burst buffer storage, HPC applications can keep compute resources busy while I/O bursts can be handled by the high-bandwidth burst buffer storage. Some typical use cases for burst buffers are data-intensive applications, in situ analysis, and checkpoint/restart [15]; the last is particularly important to ensure that systems can exhibit both high utilization and fault tolerance [14]. Previous work on simulating burst buffers examines their viability in the context of a simulated, torus-connected IBM Blue Gene/P system [16].

Further, in recent years, numerous studies have examined aspects of the incorporation and management of burst buffers [17], [18], [19], [20]. Now, HPC architectures are employing such technologies as a first class architectural component, such as National Energy Research Scientific Computing Center's (NERSC) Cori machine, supporting Cray DataWarp technology that uses SSD-based storage nodes to serve as a bridge between the compute nodes and external storage servers. Recently, DDN has announced an infinite-memory engine with burst buffers that achieves 1 TB/s of I/O performance for the Oakforest-PACS supercomputer in Japan [21]. Cray's Julia system to be deployed as part of the Human Brain Project at Jeulich Supercomputing Center will also have the DataWarp caching technology for accelerating I/O [22].

B. High-radix Interconnects

The interconnect technology of HPC systems is also undergoing major changes. While the previous generations of supercomputers (Blue Gene series [23], Cray XE and XT series [24]) often employed a torus interconnect network, the current generation systems such as Cori [1], Aurora [25], and Edison [26] have moved to a low-diameter and high-radix dragonfly network topology. Studies propose several job placement policies for the dragonfly interconnects [7], but the impact of these job placement policies on network interference is not well understood, especially with respect to the interference of communication traffic with the I/O traffic. Figure 1 shows the dragonfly architecture for the Cori system at a high level where the network routers are arranged in a two dimensional matrix format. Each row is a chassis, and the routers in the row have all-to-all connectivity. Multiple compute nodes can send read/write requests to the router having burst buffer nodes.

C. Burst Buffer Placement on High-radix Interconnects

The design of burst-buffer-enabled dragonfly systems poses several questions. First, how can burst buffers be best utilized in the presence of network contention? Second, what should be the allocation policy for these burst buffer nodes, as they will be used by multiple compute nodes in the network? Should the compute nodes select the next available burst buffer node from the available pool without considering locality? Does choosing the closest burst buffer node, for example one that is connected in the same chassis, provide better performance by causing less interference between the I/O and network traffic?

In this paper, we explore these questions by modeling the I/O workload in the form of a classic checkpoint on a simulated HPC system based on the Cray Cori architecture, where

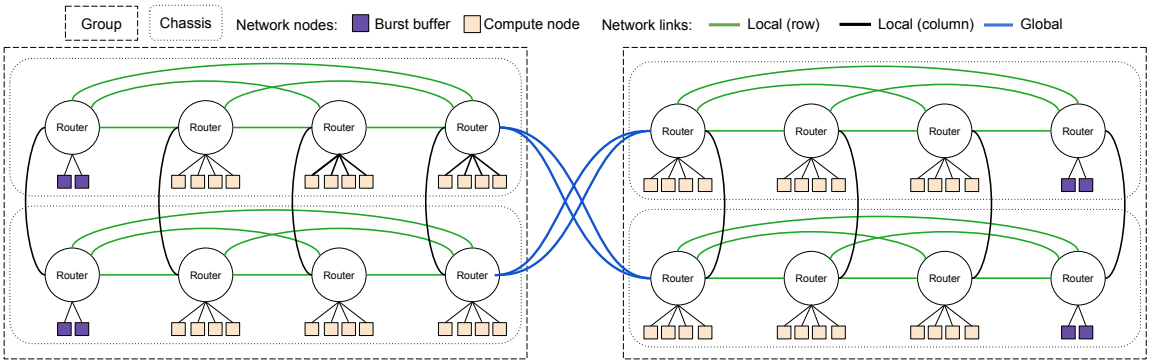


Figure 1: Burst buffer node placement on current dragonfly-based systems. Only a subset of network is shown for clarity. The evaluation done in the paper takes into account all the pieces shown.

several processes of a data-intensive application perform read/write operations on the shared burst buffer node. We study I/O and network traffic interference by using various configurations of the dragonfly routing and job placement policies along with access policies for burst buffer nodes. We examine the implications of network and I/O interference at a detailed architectural level using system configurations similar to the systems described above, as well as examining variations on the routing and resource allocation policies.

III. CODES Overview and Design

Since varying the network routing, job placement and resource allocation policies on a real HPC system would not be feasible and cost-effective, we have used the CODES simulation suite to study interference with different system configurations. CODES is a high-performance parallel discrete-event simulation framework targeting large-scale networking and storage systems relevant to HPC environments. It provides a library of validated submodels for key system components, a mechanism for combining diverse submodels into a coherent single system model, and a hierarchical configuration system that provides runtime control of the overall topology. It leverages Rensselaer's Optimistic Simulation System (ROSS) [27], [28] to scalably drive simulations. Since simulation entities are represented in ROSS as logical processes (LPs), CODES additionally provides high-level mechanisms for managing and configuring LPs from human-readable descriptions to ROSS's flat namespace. The following subsections describe each component further, summarized in Figure 2.

A. ROSS Discrete-Event Simulation Framework

ROSS provides the parallel discrete event simulation (PDES) foundation for CODES, using MPI to distribute and pass discrete timestamped messages between LPs. ROSS employs the Time Warp protocol [27], [28] to achieve scalable simulation performance for models comprising thousands or millions of LPs. Time Warp enables each LP in the model to independently and optimistically execute its local event population. Models built atop ROSS (such as

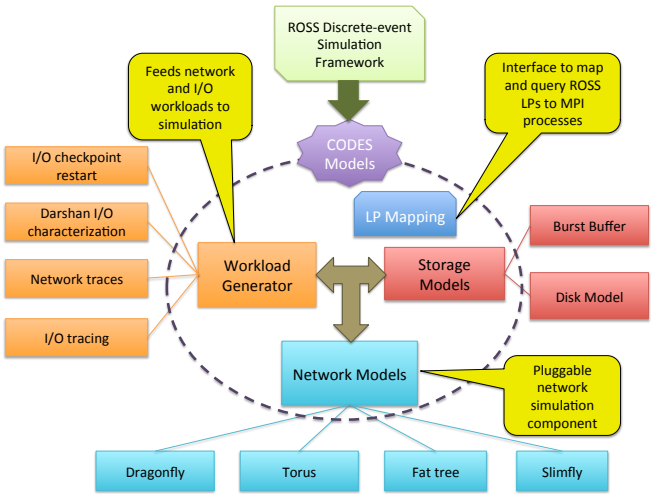


Figure 2: CODES high-level diagram.

our CODES models) provide reverse event handlers to revert the state of an LP during rollback, thereby avoiding the need for ROSS to store explicit state snapshots at every time step. This combination of optimistic coordination and lightweight rollback capability speeds up the simulations, reduces the frequency of explicit global synchronization and amount of memory capacity needed to ensure model coherence.

B. Network and Storage Components

CODES uses a simulation abstraction layer called "model-net" that allows the network models to be used as pluggable components of higher-level system or application simulations. High-fidelity (packet-level) network topologies currently modeled include the dragonfly [5], Slim Fly [29], torus [30], and fat tree [10] topologies. The model-net layer unifies common network modeling functionality such as breaking down messages into network packets and mapping simulated network endpoints to user-defined LPs. It also provides a networking API that sits on top of the network

models to simulate MPI message passing as well as RPC-style communication.

CODES provides a solid state storage (SSD) model based on the analytical techniques of Ruemmler and Wilkes [31]. The model represents SSD-based storage by scheduling read and write I/O requests in a first-come first-served manner where the available storage space can be configured by the user.

C. Workload Generator

The CODES workload generator is an abstraction layer that allows I/O and network workloads from a variety of different sources to drive the storage and network models discussed in Section III-B. The sources of network workloads currently supported are MPI application traces generated by the SST DUMPI library and synthetic network workloads such as uniform random traffic and nearest-neighbor traffic [8]. The I/O workloads come from a variety of different sources studied in previous work [32]. In the experiments presented in this paper, we use the synthetic communication pattern to represent background traffic and a checkpoint workload, discussed in Section IV-B, to represent the I/O traffic.

IV. Simulation Design

In order to study the I/O and network interference, we setup the communication and I/O workloads, the dragonfly network model and the burst buffer storage model that we are using to simulate the interference analysis.

A. Dragonfly Network Topology

The dragonfly is a hierarchical network topology having several groups connected by multiple all-to-all links. Unlike the topology originally proposed in [4], a group in the Cray XC dragonfly network [33] comprises of a fixed number of routers arranged in a two dimensional matrix. Routers in the same row and same column are connected to each other via all-to-all links. Each router has a specific number of nodes attached to it. Each router also has a number of global channels, which are inter-group connections through which one group connects to the other.

The CODES dragonfly network model supports multiple routing algorithms that have been proposed for the dragonfly networks, including adaptive, progressive adaptive, and minimal and nonminimal routing [4], [34]. Similar to the Cray systems, the adaptive routing supports both non-minimal routes within a group and across the groups [33]. Non-minimal route within a group is taken when congestion is detected on the minimal route and both source and destination nodes are in the same group. Global non-minimal route may be taken when source and destination nodes belong to different groups.

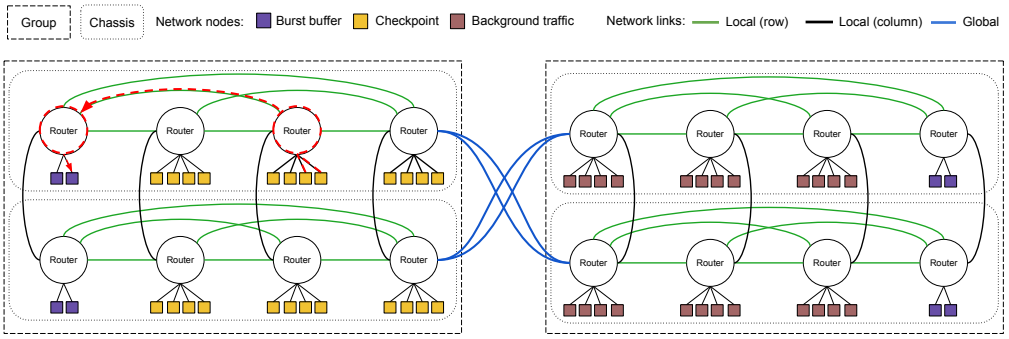
The performance measurements reported by the CODES dragonfly network model have been validated against the Theta Cray XC system at the Argonne Leadership Computing Facility (ALCF). The validation was performed by using the bisection pairing and ping-pong benchmarks [35], [36]. A variety of message sizes (ranging from 0 to 64 KiB) and network scales (up to 2,048 network nodes) were used. Performance of the CODES dragonfly model and Theta Cray XC system were similar in a majority of the cases. In the worst case, up to 8% difference was observed. More details about the dragonfly network model and its validation are presented in the report [12].

B. I/O Workload and Background Communication Traffic

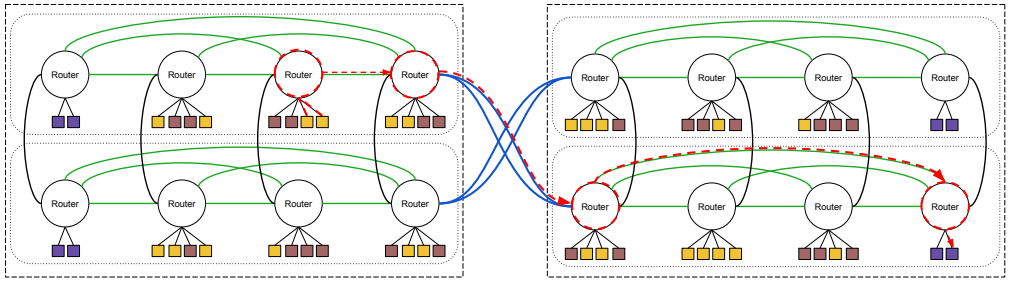
CODES provides the ability to simulate multiple jobs concurrently in the simulated system, consisting of network and/or I/O traffic, an important capability for the analyses at the center of this study. In our experiments, we execute jobs that generate I/O traffic in the form of checkpoint writes to the burst buffer nodes as well as jobs that generate uniform random network traffic, in order to explore network interference effects across independently executing workloads. Each compute node entity in the simulated system generates either checkpointing traffic or communication traffic.

1) Checkpoint Traffic: The checkpointing of application state is a widely used fault tolerance approach in HPC, but determining the frequency of checkpointing is a nontrivial task. Daly proposed an optimum checkpointing interval for applications to minimize both the time spent writing checkpoints and the time recomputing lost work due to failure, given the system's expected mean time to failure (MTTF), the amount of data to be checkpointed, and the available storage bandwidth [37]. Since checkpointing typically produces high-intensity bursts of I/O, studies indicate that utilizing burst buffers for checkpoint storage will accelerate the process [18], [19]. Additionally, using burst buffers hides the true cost of data movement to the external file system. For our experiments, we evaluate the performance of burst buffer nodes on HPC networks through a checkpointing workload based on Daly's model, implemented as a new module for the CODES workload generator (see Section III-C).

2) Background Traffic: To model interference with the checkpointing I/O traffic, we generate "background" network traffic through a separate job on the remaining non-checkpointing nodes. To form a realistic representation of the background traffic, we analyzed application traces of the HPCG [38] benchmark captured on a current generation computing platform to get the data transfer rate per rank. We configured our simulation with the same communication volume for generating the background traffic, using a uniform random pattern in which each participant randomly selects peers within the same job to receive messages. Specifically, each participant transmits a 1 KiB message every 500 microseconds. The uniform random pattern has frequently been used to evaluate high-radix HPC interconnects [4], [39] and it also represents a sub-pattern of HPC workloads such as graph computations and linear algebraic solvers [40]. Since we wanted to minimize intra-job interference, uniform random traffic was a good candidate as it is a load-balanced pattern and performs well on high-radix networks. Applica-



(a) Contiguous job allocation, nearest burst buffer allocation



(b) Random job allocation, random burst buffer allocation

Figure 3: Dragonfly topology examples with varying job and burst buffer allocation strategies. Network routes with minimal routing are shown in red. Only a subset of the network is shown for clarity.

tion pattern specific network interference has been explored by Yang et al. [9].

To warm up the simulation, background traffic starts a few simulated seconds prior to the checkpoint workload and continues until the completion of the checkpoint workload, after which a notification is sent to the background traffic generators to terminate. Note that the total quantity of background traffic depends on the checkpointing job's progress.

C. Modeling Burst Buffers

The CODES HPC system model with burst buffers is inspired by the design of the Cori HPC system at NERSC, which has specialized storage nodes as burst buffers with SSD devices installed. These burst buffer nodes are part of the interconnect network so that the compute nodes can efficiently transfer data to these nodes. Each burst buffer node on Cori has a capacity of 6.4 TB and a read/write bandwidth of 5.7 GiB/s [1]. Burst buffer nodes communicate with external storage (i.e., the parallel file system) through separate I/O nodes.

Similar to the Cori architecture, our burst buffer model occupies a subset of nodes in the dragonfly such that one router in each chassis is directly connected to two burst buffer nodes. The burst buffer model is composed of two LPs: a storage manager LP and a SSD LP. The storage manager LP manages I/O to/from the burst buffer, while the SSD LP represents the SSD storage available on the burst buffer node, as described in Section III-B. Figure 4 shows the interaction between the workload-executing compute node LPs and burst buffers. The storage manager LP receives write requests from compute node LPs and makes a blocking (with respect to the request) call to the SSD LP in order to reserve the requested space. Once space is reserved on the SSD LP, the storage manager begins a series of concurrent, pipelined RDMA reads of fixed-sized blocks from the requester and data writes to the SSD. Concurrency control on the storage manger is controlled by the fixed-at-startup memory pool allocated for block transfers and the per-operation pipelining factor.

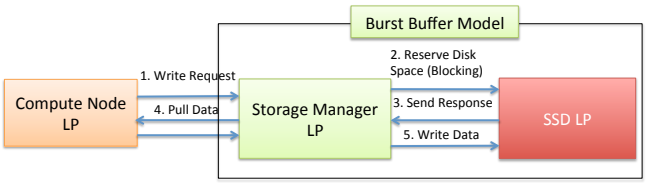


Figure 4: Interaction between compute node, storage manager, and SSD LPs

V. Evaluation

In this section, we evaluate the parameter space of job placement and burst buffer allocation policies on an assortment of job sizes to study the impact on I/O and network traffic interference. First, we present the job placement and

burst buffer policies being studied in the simulations, the simulation setup, and the metrics being used for the performance evaluation. We then describe the experimental results and analysis to quantify the I/O and network interference as predicted by our high-resolution HPC interconnect and storage simulation.

A. Simulation Configuration

We perform a strong scaling study by using a fixed aggregate checkpoint size of 1 TB and varying the job sizes from 512 to 8,192 nodes. In our simulation, one job generates a single checkpoint. The remaining nodes form a single job generating background traffic as described in Section IV-B2. Both I/O and network traffic get the same priority in the simulation, a practice followed on the current generation of HPC platforms.

We use the following network configuration, the node count and link bandwidths are comparable to the Cray Cori Phase II architecture at NERSC. The simulated system uses a dragonfly topology that services 9,600 nodes with 25 groups, where each group has 96 routers. The routers in a group are structured in a 6x16 matrix, where each router row is called a chassis. Of the 16 routers in a chassis, 15 of the routers have 4 compute nodes attached, and one router has two burst buffer nodes attached. The routers in each row are connected through green links with each link having a bandwidth of 5.25 GiB/s. Across the rows, routers are connected via three electrical links called black links [33]. Each of the black links have a bandwidth of 5.25 GiB/s. Figure 1 shows the network architecture used in the simulation for a small-scale dragonfly network. The compute nodes are connected to the routers with a bandwidth of 16 GiB/s. The network has a total of 300 burst buffer nodes, where each burst buffer node has 6.4 TB of capacity with a read/write bandwidth of 5.7 GiB/s.

In addition to the intragroup and compute node-router connections, each router has 4 global channels. Across the groups, routers are connected via these global channels referred to as blue links. The blue links use 12 optical cables between each group with each cable having a bandwidth of 4.69 GiB/s.

B. Burst Buffer Allocation Policies

Various strategies exist for mapping data from compute nodes onto the burst buffer nodes. The current job schedulers for burst buffers do not take the locality into consideration. For example, with Cray Data Warp Storage Services, compute nodes select the next available burst buffer node from the pool of burst buffer nodes in the network [41]. In this paper, we study the following allocation policies.

Random: Each compute node in a job is assigned a random burst buffer node (from anywhere in the system) to write data to for the duration of the job. This policy distributes I/O traffic across a broad range of burst buffer storage devices at the expense of introducing additional traffic on shared chassis and group links within the network. This bears similarity to the Memcached-based BurstMem work, which uses a hashing approach to map writes to burst buffer servers [19].

Nearest: Each compute node in a job is assigned to the nearest burst buffer node (in terms of network topology) to write data to for the duration of the job. This means that each application node writes to a burst buffer node on the same chassis in the architecture used in this study. If minimal routing is used with this nearest burst buffer allocation, the packets belonging to I/O traffic will traverse one green link and two compute node-router links, as shown in Figure 3a. This policy offers superior locality to the random policy but also restricts the amount of load balancing that is possible across burst buffer storage devices. This resembles the I/O forwarding model in the IBM Blue Gene series of systems as well as more specifically the IBIO burst buffer file system that uses a fixed node-to-node mapping [18].

For the purposes of this paper, we assume that the system provides a means for a job to identify the nearest burst buffer node to a given compute node. Note that neither the random nor nearest burst buffer allocation policy guarantees equal utilization of available burst buffer storage, however. We discuss the impact of this characteristic in Section V-G.

C. Job Placement Policies

Job placement is the process of assigning a set of compute nodes to a parallel application. A number of job placement policies have been suggested for the dragonfly interconnect. In this work, we explore the following policies that are either being used at supercomputer centers or have been shown to bring performance improvement as part of previous work [7], [9].

Contiguous: Each job is assigned a consecutive set of available nodes; first at router granularity, then at local group granularity, and finally across groups.

Random router: Each job is assigned all nodes directly linked to a set of randomly selected routers. Within the router sets, available nodes are ordered consecutively.

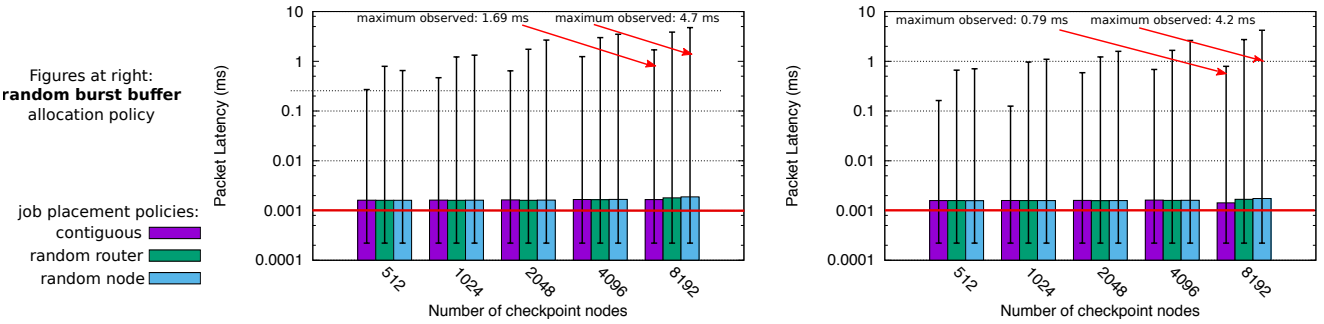
Random node: Each job is assigned a randomly selected set of nodes with no ordering constraints.

D. Performance Metrics

We use the following metrics for traffic and system analysis.

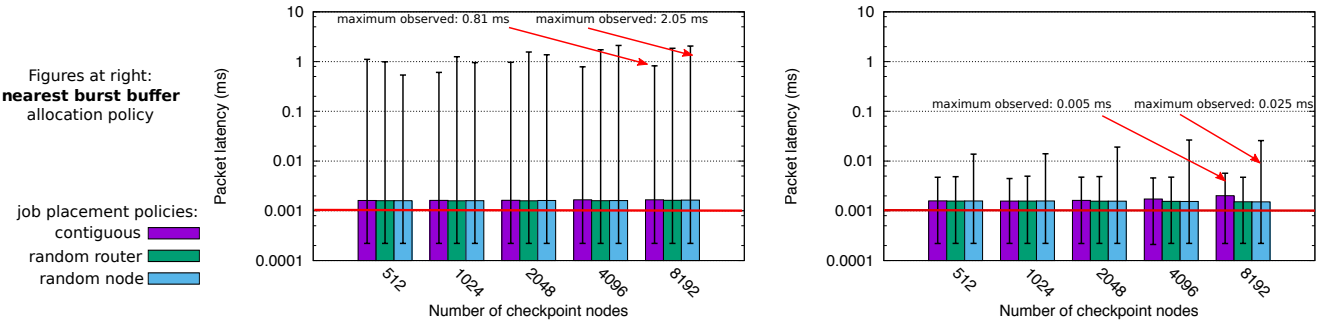
Time to complete checkpoint: This metric measures the makespan for the checkpointing job, starting when the first data transfer request is made and ending after the last data commit on any burst buffer.

Packet latency: Since the communication traffic is more latency sensitive, we use minimum, average, and maximum packet latencies to measure the performance of the background communication traffic job. The maximum packet latency is of particular relevance because of its potential impact on aggregate operations such as MPI collective routines that are sensitive to jitter.



(a) Adaptive Routing (b) Minimal Routing

Figure 5: Average packet latency observed by non-checkpoint nodes as the number of checkpoint nodes is varied from 512 to 8,192 using random burst buffer allocation. Vertical lines denote the minimum and maximum latency values in each case. Horizontal red line indicates the average packet latency without I/O traffic at 0.001ms.



(a) Adaptive Routing (b) Minimal Routing

Figure 6: Average packet latency observed by non-checkpoint nodes as the number of checkpoint nodes is varied from 512 to 8,192 using nearest burst buffer allocation. Vertical lines denote the minimum and maximum latency values in each case. Horizontal red line indicates the average packet latency without I/O traffic at 0.001ms.

E. Performance Analysis of Communication Traffic

We begin our analysis by evaluating the degree to which network communication performance is perturbed by concurrent checkpoint I/O traffic. Figure 5 shows the effect of I/O interference on the performance of background communication traffic with a randomly allocated burst buffer policy and Figure 6 with a nearest burst buffer allocation policy. We compare the slowdown in packet latency to the baseline average packet latency (0.001 ms) observed when there is no I/O traffic. We observe that although the *minimum and average* background communication packet latency shows only modest sensitivity to allocation and routing policy, the *maximum* packet latency varies significantly. This distinction is essential for applications with tightly coupled communication patterns (such as MPI collective operations) that are sensitive to jitter [42]. The major observations that we draw from these results are as follows:

Impact of burst buffer allocation scheme: The combination of nearest burst buffer allocation and minimal routing (Figure 6b) causes the *least perturbation to network communication, regardless of job allocation policy*. This is an intuitive result, because it minimizes intragroup and intergroup I/O traffic as illustrated in Figure 3a. However, even this configuration does not completely eliminate interference; the worst-case maximum background network packet latency is *5-25* times slower in the presence of checkpoint I/O traffic than without it. On the contrary, the random burst buffer allocation policy with adaptive routing (Figure 5a) produces *large network communication interference*. This is evidenced by a worst-case network packet latency of *few milliseconds*. This configuration makes the heaviest use of shared network links in the dragonfly network.

Impact of routing: Adaptive routing produces *longer worst-case latency than minimal routing* in all cases, though the difference is most pronounced with random burst buffer allocations. This is largely due to the higher-volume I/O traffic taking non-minimal routes, further exacerbating contention on shared network links. When the I/O traffic is routed minimally, the perturbation to background communication traffic is less, even with randomly selected burst buffer allocations. However, the routing policy has only a minor effect on average packet latency.

In general we see that random burst buffer allocations tend

to perturb network performance more than nearest (locality-aware) burst buffer allocations because it introduces a high volume of traffic on both group and global network links. On the other hand, with nearest burst buffer node selection, I/O data transfers utilize one green link and two compute node-router links only, minimizing interference with network communication traffic.

Impact of job placement: We observe that the job placement policies have an impact on the perturbation caused by the I/O traffic to the background communication traffic. Contiguous job placement brings the *least perturbation to background traffic* in all cases, as opposed to random router and random node policies. On the other hand, random node job placement brings *maximum jitter*, which increases as more nodes are involved in the I/O activity, even if the aggregate I/O volume is held constant.

F. Performance Analysis of I/O Traffic

In this section we evaluate the degree to which checkpoint I/O performance is perturbed by concurrent network communication traffic. We first establish a baseline using experiments with application nodes generating checkpoint traffic in isolation without background communication traffic. The experiments were then repeated with background traffic to determine the degree of perturbation to the checkpoint traffic. Figures 7 and 8 show the time to complete the checkpointing workload with and without background traffic using different job sizes, routing protocols, job placement policies, and burst buffer allocation policies.

As noted earlier in this section, the burst buffer nodes themselves (and thus their SSD storage devices) are not necessarily evenly utilized for any given checkpoint scenario simulated in this study. The nearest burst buffer allocation policy uses only burst buffers that are located in the same chassis as the checkpointing application. The random burst buffer allocation policy chooses a random target for each checkpointing application node, which means that some burst buffers receive more data than others. However, the SSD performance, even with this imbalance, is not the bottleneck for any checkpoint workloads in this study. The job and data placement combination that writes the most data on a single SSD is the nearest burst buffer case with contiguous allocation, as shown in Figure 3a. In that case, 30 compute nodes will write a total of 60 GiB of data to a single SSD when the job size is 512 nodes. With a write bandwidth of 5.7 GiB/sec, the worst case cost of writing the data to the SSD will be 10 seconds, which is less than the makespan of the checkpoint in our simulation. Any additional overhead is caused by the network fabric.

The major observations that we draw from these results are as follows:

Impact of burst buffer allocation scheme: Smaller jobs checkpoint *faster* with a random burst buffer allocation policy (Figure 7), while larger jobs checkpoint *faster* with a nearest burst buffer allocation policy (Figure 8). The reason is that the smaller jobs benefit from dispersing I/O to a larger number of burst buffers to take advantage of additional concurrent I/O paths. Large-scale jobs, in contrast, already span multiple groups with significant I/O path concurrency and thus benefit more from improved locality instead. The 512-node nearest burst buffer jobs in Figure 8 utilize no more than 10 burst buffer nodes.

Figures 9(a) and (b) show a zoomed in view of the time to complete 1 TB of checkpoint workload by each of the 512 nodes using randomly selected and nearest burst buffer nodes, respectively. With randomly selected burst buffer, the time to complete checkpoint traffic increases continuously as the packets take non-minimal routes to avoid congestion. With nearest burst buffer selection, we see groups of network nodes completing their checkpoint process earlier than the others, which constitutes the steps in the graph. This is because the network links get fully congested, which causes some network nodes to get delayed in completing their checkpoint traffic.

Impact of routing: Adaptive routing offers a slight improvement to checkpoint traffic time for randomly allocated burst buffers. This allows I/O traffic to avoid heavily contended links, although the tradeoff (shown in the preceding section) is that it has a negative impact on background traffic on the system.

For nearest burst buffer allocation policy, adaptive routing offers *improvement* with random node and random router job placements but brings a *slow down* with contiguous job placement. With contiguous job placement, the compute node and its nearest burst buffer node lie in the same group, therefore, adaptive routing takes non-minimal routes within a group to alleviate congestion. However, since all compute nodes in a group are participating in I/O, this increases network contention and slows down the I/O traffic.

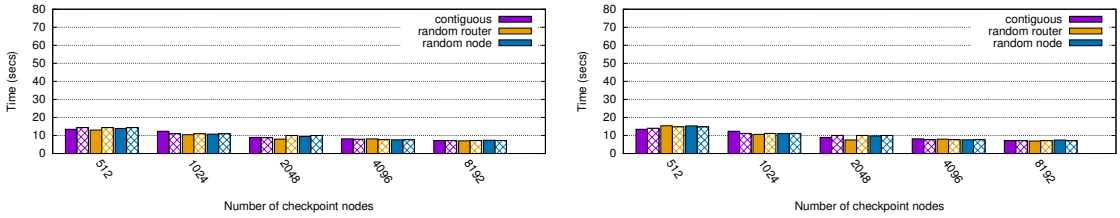
Impact of job placement: The job placement policies have a *noticeable impact with nearest burst buffer* selection policy (Figure 8). Since contiguous job placement restricts the traffic in a specific part of the network, it over-utilizes selected burst buffer nodes while the rest of the burst buffer nodes stay un-utilized. Random node and random router job placements offer the opportunity for a *better utilization of burst buffer nodes* with locality aware burst buffer placement and result in a faster time to complete I/O.

G. Discussion of Tradeoffs

We analyze the results for both background communication and I/O traffic to find the configuration that brings minimum perturbation to communication traffic and efficiently utilizes the burst buffer nodes. Our observations are as follows:

- Nearest burst buffer allocation (i.e., a topology-aware mapping of burst buffer nodes to compute nodes) is an intuitive policy for burst buffer allocation, but checkpoint performance and communication performance are directly (and inversely) impacted by the job placement policy in this configuration (Figure 8). Furthermore, the benefits of nearest buffer allocation are limited

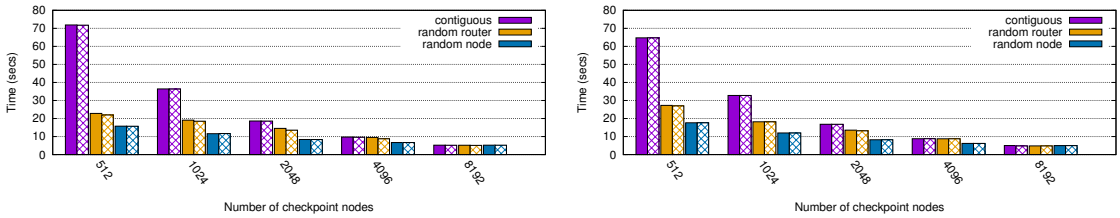
Figures at right:
random burst buffer
allocation policy



(a) Adaptive routing (b) Minimal routing

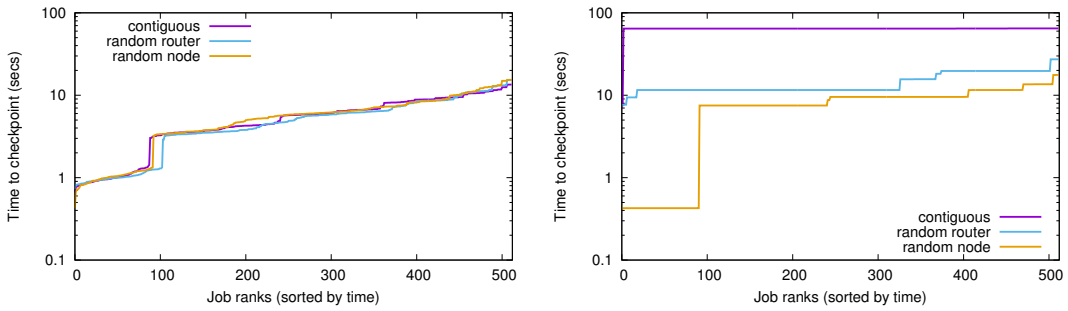
Figure 7: Time to complete checkpoint as the number of nodes participating in the checkpoint is varied from 512 to 8,192 with random burst buffer allocation. The solid color bars denote checkpoint time with background traffic while the crosshatched bars denote checkpoint time without background traffic for each job allocation strategy. The aggregate checkpoint size is held constant at 1 TB in all configurations.

Figures at right:
nearest burst buffer
allocation policy



(a) Adaptive Routing (b) Minimal Routing

Figure 8: Time to complete checkpoint as the number of nodes participating in the checkpoint is varied from 512 to 8,192 with nearest burst buffer allocation. The solid color bars denote checkpoint time with background traffic while the crosshatched bars denote checkpoint time without background traffic for each job allocation strategy. The aggregate checkpoint size is held constant at 1 TB in all configurations.



(a) Random burst buffer allocation strategy (b) Nearest burst buffer allocation strategy

Figure 9: Time to complete checkpoint by each checkpoint node for a 512 node checkpoint job with minimal routing. Time with random burst buffer placement increases from 0.4 to 16 seconds. The steps in nearest burst buffer allocation case are due to network congestion.

if the application's I/O phase requires completion by all nodes, due to the variability in completion time as shown in Figure 9(b). While contiguous placement causes the least perturbation to background traffic, it increases the I/O completion time by up to 5 times. Random router policy is 1.5 times slower than the random node policy, but it causes less perturbation to background traffic. The random node policy has a high-performing I/O completion time in all cases but causes more perturbation to the background network traffic.

- In addition to the job placement and burst buffer selection, routing plays a key role in determining the performance of I/O and network traffic. While adaptive routing takes nonminimal routes to reduce congestion, it interferes with the background traffic and causes performance degradation. Minimal routing creates congestion on the network and SSDs when used with compact job and burst buffer allocation policies, but it causes less interference with background network traffic.

- *Overall, in order to attain minimum perturbation to background network traffic and effectively utilizing burst buffers for I/O traffic, the I/O traffic should be routed minimally while using the closest burst buffer nodes* (Figures 8b and 6b). To alleviate congestion introduced by the I/O traffic, the *jobs should be placed by using either random node or random router* strategies. The tradeoff is that random node job placement would introduce more perturbation to the background traffic while random router placement would impact the I/O performance.

VI. Conclusion and Future Work

The addition of a burst buffer storage tier and the recent shift in interconnect topology of HPC systems have created several questions for the research community. In this work, we have applied the CODES simulation framework, based on composable HPC-oriented modules, to quantify the I/O and network traffic interference using various configurations of job sizes, job placement policies, routing and burst buffer allocation policies on dragonfly networks. Our analyses used detailed packet-level simulations to do performance predictions for 1 TB of checkpoint I/O traffic in the presence of background communication traffic.

From the communication traffic perspective, we observed that configurations with non-topology aware burst buffer assignments brought a wide variation in packet latencies of the background communication traffic. In the worst-case a packet latency of 4700 times the average latency has been recorded. Configurations with a topology-aware mapping of burst buffer nodes to compute nodes offered minimum variation in packet latencies but slowed the I/O traffic by creating contention on the burst buffer nodes. Balancing the tradeoff between I/O and network performance required choosing an optimal configuration of job placement, burst buffer allocation, and routing policies.

The paper aims to establish a baseline methodology for quantifying I/O and communication traffic with different configurations of a high-radix dragonfly network. The work can be extended by studying a wider variety of workloads and other HPC interconnect topologies. We have made the simulation framework open source, and the mechanisms for workloads and interconnect models are modular and can be easily extended.

Appendix

A. Simulation Suite Description

The CODES, ROSS and codes-storage-server are open source toolkits available for HPC interconnect and storage design space exploration. The source code for the burst buffer storage model and workload replay is available at https://xgitlab.cels.anl.gov/codes/codes-storage-server.git. The job allocation files and network configuration files used in the experiments are also available in the source repo. Documentation need to run the experiments in this paper is provided at https://xgitlab.cels.anl.gov/codes/codes-storage-server/wikis/checkpoint-study.

Acknowledgment

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC02-06CH11357 and Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-731482). The work used resources at the Argonne Leadership Computing Facility (ALCF) and Rensselaer's CCI supercomputing center. We gratefully acknowledge Xu Yang (IIT) and Xin Wang (IIT) for their help with the CODES network models.

References

[1] NERSC, "*Cori*," https://www.nersc.gov/users/computational-systems/cori/.

[2] ——, "*Trinity and NERSC-8 Computing Platforms: Draft Technical Requirements*," https://www.lanl.gov/business/vendors/_assets/docs/Trinity-NERSC-8-DRAFT-technical-requirements.pdf.

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

[4] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," *ACM SIGARCH Comput. Architecture News*, vol. 36, no. 3, pp. 77–88, Jun. 2008.

[5] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns, "Modeling a million-node dragonfly network using massively parallel discrete-event simulation," in *High Performance Comput., Networking, Storage and Anal. (SCC) SC Companion*, 2012, pp. 366–376.

[6] B. Prisacari, G. Rodriguez, P. Heidelberger, D. Chen, C. Minkenberg, and T. Hoefler, "Efficient Task Placement and Routing of Nearest Neighbor Exchanges in Dragonfly Networks," in *HPDC 2014*. New York, NY, USA: ACM, 2014, pp. 129–140.

[7] N. Jain, A. Bhatele, X. Ni, N. J. Wright, and L. V. Kale, "Maximizing throughput on a dragonfly network," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 336–347.

[8] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns, "Enabling parallel simulation of large-scale HPC network systems," in *IEEE Transactions on Parallel and Distributed Systems*. IEEE, 2016.

[9] X. Yang, J. Jenkins, M. Mubarak, R. Ross, and Z. Lan, "Watch out for the bully! job interference study on dragonfly networks," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2016.

[10] N. Jain, A. Bhatele, S. T. White, T. Gamblin, and L. V. Kale, "Evaluating HPC setworks via simulation of parallel workloads," in *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking,*

Storage and Analysis, ser. SC '16. IEEE Computer Society, Nov. 2016, lLNL-CONF-690662.

[11] Argonne Leadership Computing Facility (ALCF), "*Theta, Argonne's Cray XC System*." [Online]. Available: https://www.alcf.anl.gov/theta

[12] Mubarak, Misbah and Ross, Robert B. and Carothers, Christopher D., "*Validation of CODES dragonfly model with Theta Cray XC System*," 2017. [Online]. Available: http://www.mcs.anl.gov/publication/validation-study-codes-dragonfly-network-model-theta-cray-xc-system

[13] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock, "I/O Performance Challenges at Leadership Scale," in *Supercomputing (SC) 2009*, ser. SC '09. New York, NY, USA: ACM, 2009, pp. 40:1–40:12.

[14] D. Zhao, D. Zhang, K. Wang, and I. Raicu, "Exploring Reliability of Exascale Systems Through Simulations," in *Proceedings of the High Performance Computing Symposium*, ser. HPC '13. San Diego, CA, USA: Society for Computer Simulation International, 2013, pp. 1:1–1:9.

[15] D. Bard, "Accelerating science with the NERSC burst buffer early user program." [Online]. Available: http://salishan.ahsc-nm.org/uploads/4/9/7/0/49704495/2016-bard.pdf

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

[17] 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*, ser. ROSS '12. ACM, 2012, pp. 4:1–4:6.

[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 *CCGrid 2014*, May 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*, Oct. 2014, pp. 71–79.

[20] 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 *HPDC 2016*, ser. HPDC '16. New York, NY, USA: ACM, 2016, pp. 69–80.

[21] DDN, "DDN infinite memory engine." [Online]. Available: http://www.ddn.com/press-releases/ddn-infinite-memory-engine-burst-buffer-exceeds-1-tb-per-second-file-system-performance-for-japans-fastest-supercomputer/

[22] H. B. Project, "Juelich Supercomputing Center." [Online]. Available: https://top500.org/news/juelich-supercomputing-centre-deploys-cray-and-ibm-supercomputers-for-human-brain-project/

[23] N. R. Adiga *et al.*, "Blue Gene/L torus interconnection network," *IBM J. of Res. and Develop.*, vol. 49, no. 2.3, pp. 265–276, Mar. 2005.

[24] "NERSC Hopper retired supercomputer," Online: http://www.nersc.gov/users/computational-systems/retired-systems/hopper/, last visited Sept. 30, 2015.

[25] Argonne Leadership Computing Facility (ALCF), "*Aurora, Argonne's next generation supercomputer*." [Online]. Available: http://aurora.alcf.anl.gov(Accessedon:Apr.27,2015)

[26] "NERSC Edison supercomputer," Online: http://www.nersc.gov/users/computational-systems/edison/, last visited Sept. 30, 2015.

[27] C. D. Carothers, D. Bauer, and S. Pearce, "ROSS: A high-performance, low-memory, modular Time Warp system," *J. of Parallel and Distributed Comput.*, vol. 62, no. 11, pp. 1648–1669, Nov. 2002.

[28] P. D. Barnes, C. D. Carothers, D. R. Jefferson, and J. M. LaPre, "Warp speed: Executing Time Warp on 1,966,080 cores," in *Proc. of the 2013 ACM SIGSIM Conf. on Principles of Advanced Discrete Simulation (PADS)*, May 2013, pp. 327–336.

[29] N. Wolfe, C. D. Carothers, M. Mubarak, R. Ross, and P. Carns, "Modeling a Million-Node Slim Fly Network Using Parallel Discrete-Event Simulation," in *SIGSIM PADS 2016*. New York, NY, USA: ACM, 2016, pp. 189–199.

[30] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns, "A case study in using massively parallel simulation for extreme-scale torus network codesign," in *Proc. of the 2nd ACM SIGSIM/PADS Conf. on Principles of Advanced Discrete Simulation*, 2014, pp. 27–38.

[31] C. Ruemmler and J. Wilkes, "An Introduction to Disk Drive Modeling," *Computer*, vol. 27, no. 3, pp. 17–28, Mar. 1994.

[32] S. Snyder, P. Carns, R. Latham, M. Mubarak, R. Ross, C. Carothers, B. Behzad, H. V. T. Luu, S. Byna, and Prabhat, "Techniques for Modeling Large-scale HPC I/O Workloads," in *PMBS workshop at SC'15*. ACM, 2015, pp. 5:1–5:11.

[33] G. Faanes *et al.*, "Cray cascade: a scalable HPC system based on a dragonfly network," in *Proc. of the Int. Conf. on High Performance Comput., Networking, Storage and Anal. (SC)*, 2012, pp. 103–113.

[34] J. Won, G. Kim, J. Kim, T. Jiang, M. Parker, and S. Scott, "Overcoming far-end congestion in large-scale networks," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 2015, pp. 415–427.

[35] R. Hockney, "Performance parameters and benchmarking of supercomputers," *Parallel computing*, vol. 17, no. 10-11, pp. 1111–1130, 1991.

[36] W. Gropp and E. Lusk, "Reproducible measurements of MPI performance characteristics," in *Proc. of the 6th Eur. PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Mach. and Message Passing Interface*, 1999, pp. 11–18.

[37] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Generation Computer Systems*, vol. 22, no. 3, pp. 303–312, 2006.

[38] NERSC, "*HPCG benchmark*," http://www.nersc.gov/research-and-development/apex/apex-benchmarks/hpcg/.

[39] M. Besta and T. Hoefler, "Slim Fly: A cost effective low-diameter network topology," in *Proc. of the Int. Conf. for High Performance Comput., Networking, Storage and Anal. (SC)*, 2014, pp. 348–359.

[40] X. Yuan, S. Mahapatra, W. Nienaber, S. Pakin, and M. Lang,

"A new routing scheme for Jellyfish and its performance with HPC workloads," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 36.

[41] M. Gilge, "Cray Data Warp user's guide." [Online]. Available: http://docs.cray.com/books/S-2558-5204/S-2558-5204.pdf

[42] F. Petrini, D. J. Kerbyson, and S. Pakin, "The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q," in *Supercomputing, 2003 ACM/IEEE Conference*. IEEE, 2003, pp. 55–55.