Evaluating HPC Networks via Simulation of Parallel Workloads

Nikhil Jain[†,⋆], Abhinav Bhatele[†], Sam White[⋆], Todd Gamblin[†], Laxmikant V. Kale[⋆]

[†]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, California 94551
[⋆]Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801
E-mail: [†]{nikhil, bhatele, tgamblin}@llnl.gov, [⋆]{white67, kale}@illinois.edu

***Abstract*—This paper presents an evaluation and comparison of three topologies that are popular for building interconnection networks in large-scale supercomputers: torus, fat-tree, and dragonfly. To perform this evaluation, we propose a comprehensive methodology and present a scalable packet-level network simulator, TraceR. Our methodology includes design of prototype systems that are being evaluated, use of proxy applications to determine computation and communication load, simulating individual proxy applications and multi-job workloads, and computing aggregated performance metrics. Using the proposed methodology, prototype systems based on torus, fat-tree, and dragonfly networks with up to 730K endpoints (MPI processes) executed on 46K nodes are compared in the context of multi-job workloads from capability and capacity systems. For the 180 Petaflop/s prototype systems simulated in this paper, we show that different topologies are superior in different scenarios, i.e. there is no single best topology, and the characteristics of parallel workloads determine the optimal choice.**

***Index Terms*—Multiprocessor interconnection networks, Network topology, Computer simulation, Performance evaluation, High performance computing**

I. Introduction and Motivation

The interconnection network of a supercomputer plays an important role in determining its overall efficacy. Communication-intensive applications can achieve high scalability only if a suitable, capable network is available, and if those applications are able to exploit it efficiently. The inability to scale communication to high node counts can prevent the use of more compute nodes, undermining the benefit of building large-scale systems. Hence, detailed performance comparisons of different networks should be conducted to identify the best and most suitable option for a particular computing center's expected workload.

Traditionally, analytical modeling has been used to decide the best choice of networks and routing schemes [1], [2], [3], [4], [5], [6], [7], [8]. However, these models make simplifying assumptions about communication flow and congestion on networks in order to be fast, thus leading to potentially low-accuracy predictions. Alternatively, sequential flit- and packet-level simulations have been used for higher-accuracy performance predictions [9], [10], [11], [12]. This approach can result in significantly high execution times for large simulations, and can render the simulation of full, realistic workloads infeasible.

Past work has used aggregate metrics such as average packet latency and throughput to compare networks and routing schemes [8], [13], [14], [15], [16], [17]. In these studies, synthetic workloads parameterized on injection rate, message size, and message destination are used. These metrics and synthetic workloads fail to capture scenarios of practical importance to high performance computing (HPC). On production HPC systems, one or many applications with communication-computation and communication-communication dependencies are executed as multi-job workloads. Complexities of diverse scenarios found in such workloads cannot be replicated using simple synthetic workloads.

This paper presents a comprehensive methodology to evaluate interconnection networks that addresses the aforementioned shortcomings. We propose that performance prediction based on parallel packet-level simulations of diverse multi-job workloads representing production HPC applications provides meaningful insights about the efficacy of different networks for HPC centers. Our work is motivated by three key software developments in the HPC community.

First, the ROSS Parallel Discrete Event Simulation (PDES) framework [18], [19] provides scalable parallelization of fine-grained discrete event simulations. This has enabled development of parallel packet-level network simulators that can scale to thousands of cores as demonstrated by the CODES framework [20], [21], [22].

The second development is connecting BigSim's emulation framework [10] with CODES via TraceR [23]. This not only enables replay of control and communication flow of MPI [24] and Charm++ [25] applications, but also makes it feasible to include and model computation time in simulations. Moreover, virtualization in BigSim's emulation framework enables simulation of control and communication flows of prototype systems much larger than the system running the emulation. Fig. 1 shows that simulations performed using TraceR achieve good strong scaling as the number of simulating cores is increased, while allowing emulation on a few tens of cores.

Finally, representative proxy applications for several production applications have been developed recently as part of efforts such as the Mantevo project [26], the CORAL collaboration [27], and NERSC-8/Trinity benchmarks [28]. These proxy applications provide an easy-to-use, customizable way to capture and replay control and communication flow of multi-job workloads of interest to the HPC community.

In this work, we augment both CODES and TraceR with new capabilities, and deploy them to evaluate network topolo-

SC16; Salt Lake City, Utah, USA; November 2016
978-1-4673-8815-3/16/$31.00

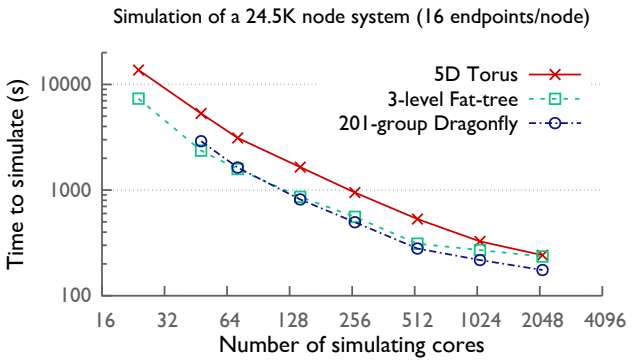

Fig. 1. Excellent strong scaling performance of ROSS enables massive workload simulations using TraceR and CODES.

gies that are popular for building interconnects in large-scale supercomputers. The main contributions of this paper are:

- We demonstrate the capabilities of TraceR to efficiently simulate complex HPC workloads containing multiple jobs for large-scale systems using different job placements and task mappings.
- We have added dynamic routing to the torus model and a new, full bisection bandwidth, fat-tree network model in CODES.
- We present a comprehensive methodology for comparing performance of different network topologies to build future HPC systems.
- We evaluate networks based on torus, fat-tree, and dragonfly topologies using realistic workloads from capability and capacity systems. To the best of our knowledge, this is the first study that compares HPC networks for such complex workloads.

II. Augmentations to CODES and TraceR

In order to conduct studies on different networks using multi-job workloads as described in Section III, several capabilities have been added to CODES and TraceR.

A. CODES

The CODES framework [20] is built upon ROSS [18] to facilitate studies of HPC storage and network systems. CODES allows instantiation of prototype networks based on packet-level models defined in it. These instantiations are guided by parameters such as network type, dimensions, link bandwidth, switch latency, etc. At the beginning of this project, CODES provided n-d torus [29] and dragonfly [13] network models among others. The following improvements were made to the network models in CODES as part of this work.

Deadlock avoidance mechanisms: A packet-level token flow control mechanism is used in CODES to manage flow of traffic. It can cause deadlocks if cyclic dependencies are formed among the transmitted packets. To avoid deadlocks in the torus model, we added Puente et al.'s *bubble* routing scheme that uses an *escape* channel [30]. In the dragonfly model, deadlocks have been avoided by using as many virtual channels per link as the maximum number of hops a packet may take as proposed by Kim et al. [13].

Dynamic routing in the torus model: The existing torus model in CODES supports dimension-ordered static routing only [21]. However, it has been shown that adaptive or dynamic routing improves communication performance on torus, especially for applications with large messages [31], [32], [33]. Hence, we added a fully-adaptive minimal path routing scheme to the torus network model.

Based on the routing scheme used in the torus networks of IBM's Blue Gene series [33], the implemented adaptive routing scheme moves packets towards their destination by making locally optimal decisions. When a packet arrives at an intermediate router, it is forwarded to the port with minimum occupancy of the buffers among the ports that move the packet closer to its destination.

Fat-tree network model: We have added a full bisection fat-tree network model with up to three levels of switches to CODES. In our construction of fat-tree network using radix k switches, sets of $\frac{k}{2}$ switches are grouped together to form *pods* at the leaf and intermediate levels. Switches from corresponding pods at the leaf and intermediate levels are connected in an all-to-all fashion. Many switches of same radix as the lower layers are used to form logical high radix switches at the topmost layer as described in [34].

This mode of building fat-tree networks resembles a folded Clos network [35], and is often used for practical deployment, e.g. in Cab at Lawrence Livermore National Laboratory, and in simulations [9], [36]. Given radix k switches and n levels, the implemented fat-tree network model supports up to $\frac{k}{2}$ pods and $(\frac{k}{2})^n$ nodes. Alternate constructions of fat-tree which can support k or $2k$ pods will be explored in a future work.

For the fat-tree network, several pattern specific static routing schemes have been proposed [37], [38]. Given our focus on comparison of different networks using a diverse range of benchmarks, we have implemented a pattern-oblivious shortest-path adaptive routing scheme. In this scheme, when a packet arrives at a switch, all ports through which the packet can be forwarded are computed. Among these ports, the port with minimum occupancy of buffer is selected to enqueue the packet. We acknowledge that this routing scheme may not be the best routing scheme for certain communication patterns, but it provides a fair comparison of fat-tree networks with other networks for the general case.

B. TraceR

TraceR is a successor to BigSim's application simulator [10], [39] that replays control and communication flow on top of CODES [23]. The BigSim project is aimed at developing tools that allow prediction and tuning of the performance of applications before next generation machines are available in production. The primary advantage of BigSim is its ability to emulate and generate traces for large number of endpoints (entities that initiate communication) using systems with relatively small number of cores. We have added the following capabilities to TraceR as part of this work:

Task mapping: The ability to map endpoints at runtime can be critical to obtaining the best network performance [40], [41], [8]. To enable topology-aware mapping in simulation, we extended TraceR to accept a *mapping* file that determines the placements of endpoints identified by their ranks to cores in the prototype system being simulated. This allows TraceR to more accurately reflect real world behavior.

Simulation configurations: Emulation of large-scale executions to generate traces can be cumbersome, especially when application executions with many different message sizes and computation times are to be emulated. To ease exploration of such design space, we added the capability to change message sizes and projected execution time during simulation. Users can mark messages and computation tasks during emulation with names of their choice, and then modify their values during simulation via configuration input to TraceR. Additionally, one can generate traces for only one iteration of application execution, and let TraceR iterate over that work multiple times during simulation.

Multi-job simulations and placement: Realistic workloads in HPC include several applications of various sizes executing simultaneously. Evaluating such scenarios is critical to determining the efficacy of different network for practical use cases. To this end, TraceR has been augmented with the capability to simulate many jobs executed simultaneously on prototype systems. Different applications can be emulated independently to generate traces. During simulation, description of individual jobs (including the job size, location of traces, etc.) and their placement on the network is provided to TraceR, which then simulates them simultaneously. Within each job, task mapping and simulation configurations can be set. Such multi-job simulations with functionalities described above are not possible in any other simulator.

Fig. 2 summarizes all input configurations that can be specified to TraceR and the output obtained from the simulations.

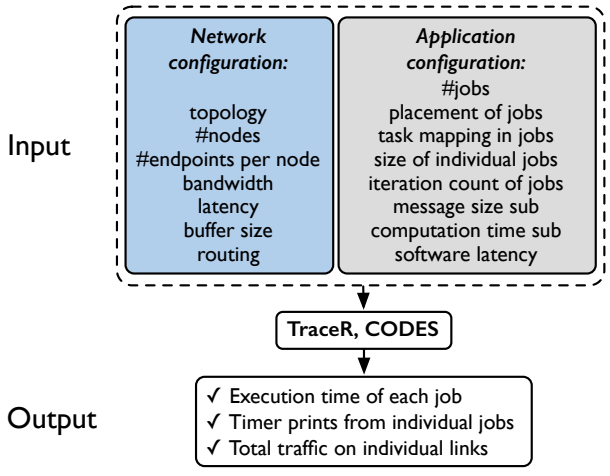

Fig. 2. Various configurable input parameters (w.r.t. the network model and application traces) and output of TraceR.

Simulation validation: The torus and dragonfly network models have been extensively validated [20], [23]. For all the network models, we have executed controlled micro-benchmarks, e.g. ping-pong, and have ensured by manual inspection that at every step, the simulation behaves as expected. The torus model has been further validated by comparing simulation results from TraceR with executions on a IBM Blue Gene/Q system for three benchmarks: pair-wise ping-pong, multi-pair ping-pong, and a permutation pattern in which every MPI rank communicates with its diagonally opposite MPI rank. For both static and dynamic routing, predictions within 2% of real execution time were observed. The fat-tree model has been validated by comparing observed latency and throughput from TraceR with BookSim's prediction [9]. It has also been validated by comparing predictions from TraceR with runs of the three aforementioned benchmarks on Catalyst, a system with fat-tree network.

III. Evaluation Methodology

We follow a four step methodology that enables comprehensive evaluation and comparison of different networks.

1) Prototype system design: The first step for evaluating a network is designing a prototype system that can be instantiated in CODES. As shown in Fig. 2, this requires selecting many parameters that are part of the network configuration provided as input to TraceR. Section III-A describes in detail the significance of these options, and the choices that have been made for results presented in this paper.

2) Workload selection: Next, we select a set of benchmarks and proxy applications that are simulated on the systems. The choices made here depend primarily on the computation and communication patterns that are expected to be run on the systems. Recent focus of the HPC community on developing proxy applications helps in this step since it provides easy-to-use, customizable codes that represent diverse applications executed on production systems [26], [27], [28]. We have chosen four benchmarks and two proxy applications that represent and span a wide-range of application design space in HPC:

- Stencil: near-neighbor communication in 3D with relaxation updates [42], [43], [44].
- Transpose: permutation communication with matrix-matrix multiplication [45], [46].
- FFT: all-to-all messages with 1D fast Fourier transform computation [47], [48], [46].
- Fine-Grained: small messages to many neighbors with property updates [49], [50].
- Kripke: discrete-ordinates (S_N) transport code with sweeps and critical path messaging [51].
- Qbox-Mini: multiple MPI collectives in first principles molecular dynamics [52].

Using these proxy applications, we create workloads that represent three different modes in which supercomputers are used in general: a) *Single jobs*: only one application is executed on the entire system, b) *Capability workloads*: a few large-sized applications are executed simultaneously, and c) *Capacity workloads*: many jobs with variable sizes are executed on the system simultaneously.

TABLE I
DESIGN CHOICES FOR PROTOTYPE SYSTEMS. SPECIFIC VALUES SHOW THE CONFIGURATIONS SIMULATED AND COMPARED IN THIS PAPER.

Design Choice	Torus	Dragonfly	Fat-tree
Number of nodes (n)	~46656	~46656	~46656
Router radix (r)	13	$6 \times \text{ceil}\left(\left(\frac{n}{18}\right)^{\frac{n}{4}}\right) = 48$	$2 \times \text{ceil}\left(n^{\frac{1}{3}}\right) = 72$
Nodes per router	1	$r/6 = 8$	$r/2 = 36$
Routing scheme	Shortest-path adaptive	Adaptive hybrid	Shortest-path adaptive
Network specifics	6D torus: $6\times6\times6\times6\times6\times6$	240 groups with 24 routers each	3-level with full bisection

3) Emulation and simulation: After creating the workloads, we emulate each of the proxy applications with different job-sizes using BigSim's emulation framework to generate traces that drive the simulation. During emulation, we tag computation routines and message sends with unique names so that their characteristics, e.g. time to perform computations and size of the messages, can be changed during simulation.

To conduct simulations using TraceR, different job placements and task mappings are created in order to optimize execution on each prototype system. For every proxy application, computation time and message sizes used in the simulation have been decided based on problem sizes that are typically used for them at a given scale and using the benchmarking information available from the CORAL and Trinity procurement websites [27], [28]. Computation time is obtained by benchmarking the proxy applications on current systems and projecting those values based on the expected flop/s provided by prototype systems. The simulation results thus obtained provide the performance metrics that are used for evaluating the prototype systems.

4) Combining aggregated performances: The last step in our methodology is to compute aggregated performance metrics for high level comparison of different networks. Section VII provides details related to this step.

A. Prototype System Design

We focus on three topologies that are widely used in production HPC interconnects (torus, fat-tree, and dragonfly) to build prototype systems of similar capabilities. Since the general trend in HPC is to measure a system's capability by the peak flop/s it provides, the prototype systems we build for comparison also have similar peak flop/s. Further, to eliminate the effect of variable flop/s per node, we assume that nodes with 4 TF peak capacity are used in each prototype system. The end result is that we compare different network topologies assuming similar node counts. This also implies that for our prototype systems to have compute capacity similar to next generation supercomputers ($\sim$180 PF), the prototype systems have $\sim$46 K nodes.

In order to primarily focus on the effects of interconnect topology, we keep the following design choices constant across different prototype systems: bandwidth of links connecting a pair of routers (12.5 GB/s), injection bandwidth of bus/link that connects a node's NIC to its router (12.5 GB/s or 25 GB/s), size of packets that are created by the NIC (1,024 bytes), and buffer size of the virtual channels that store packets in transit (64 packets). Two variations of prototype systems are simulated for all topologies: one in which injection bandwidth between the NIC and the router is same as the link bandwidth - 12.5 GB/s (called BALANCED INJECTION), and the other in which the injection bandwidth is $2\times$ the link bandwidth - 25 GB/s (called HIGH INJECTION). This has been done to prevent the injection bandwidth from being the bottleneck for configurations with imbalanced communication load.

Design choices made based on the network topology include the connectivity of routers, number of nodes per router, and available routing schemes. The number of nodes per router is selected so that each network is balanced, i.e. when uniformly distributed traffic is generated from all nodes, routers should be able to forward packets without delays in an ideal scenario. Table I summarizes the topology dependent design choices made in our prototype systems that are described here for each of the topology.

Torus: A generic n-dimensional torus can be viewed as an extension of a n-dimensional grid in which the edges are wrapped around to form rings. Torus of various dimension (two to six) have been used in many supercomputers in the last decade [29], [53], [54], [55]. In a typical torus network, messages travel many hops when sent from a source node to a destination node. Thus, the best one can do to create a balanced torus-based system is to attach only one node per router [56]. Since the bisection bandwidth typically increases as the number of dimensions are increased, our prototype system based on torus uses a 6D torus, which is the largest dimensionality that has been used in a production torus-based system [53]. The $46,656$ nodes are arranged as $6 \times 6 \times 6 \times 6 \times 6 \times 6$.

Dragonfly: Dragonfly is a multi-level dense topology aimed at making use of high-radix routers [13], [57]. A typical dragonfly consists of two levels of connections. At the first level, routers are connected via a dense topology, e.g. an all-to-all, to form groups. The groups behave as virtual routers which are in turn connected in an all-to-all manner. Dragonfly and its variations have been used in recent supercomputer networks [58], [57], and is expected to be the network topology of Aurora [59].

Kim et al. [13] proposed that for a dragonfly network built using routers with radix k, $\frac{k}{4}$ ports should be connected to compute nodes, $\frac{k}{2}$ ports should be used for local connections within a group, and $\frac{k}{4}$ connections should be used for global connections across groups. These values are suitable for creating a balanced system which uses direct routing, i.e., for every packet injected onto the network, two local and one

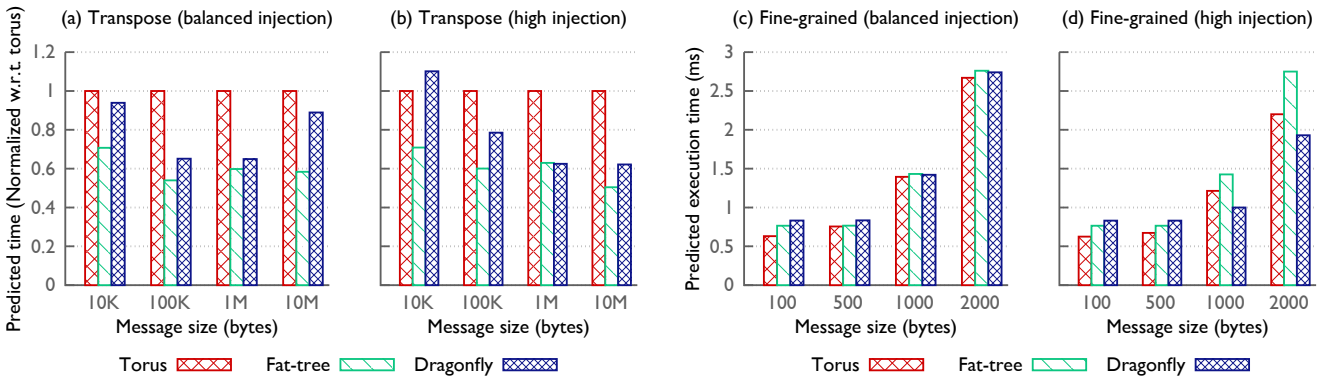

Fig. 3. **TRANSPOSE**: The fat-tree machine provides the best performance, while the torus machine has the worst performance in most cases. **FINE-GRAINED**: The torus machine provides the best performance with balanced injection; the dragonfly machine gets the most benefit from having higher injection bandwidth.

global links are used in the worst case.

However, recent studies and practical deployments [8], [60], [58] have shown that indirect or hybrid routing is essential to obtain good performance on dragonfly networks. With indirect routing, three local links and two global links may be used for a given packet. Thus, in a balanced system which uses indirect routing, $\frac{k}{6}$ ports should be connected to nodes, $\frac{k}{2}$ ports should be used for local connections, and $\frac{k}{3}$ ports should be used for global connections. These values are used in our prototype system that consists of 240 groups each with 24 routers.

For the dragonfly-based prototype system, loop-back links are left unused for intra-group connections for ease of simulations with CODES. Further, since a full-sized system built using 48 port routers has ~73K nodes, we use only ten global links per router (instead of 16) and increase their bandwidth by $1.6\times$ to restore the balance of the system.

Fat-tree: The key idea behind the fat-tree network is the following: in a tree topology, the loads on the links increase as we approach the root; thus, link bandwidth should be higher for the links closer to the root [61]. Practical deployments of fat-tree replace the fat switches near the root with many smaller switches that logically behave as one large switch as described in Section II-A. This topology has been widely adopted for deployment of Infiniband-based supercomputers [62], [63]. Fat-tree is also expected to be used in two of the largest next generation supercomputers, Summit [64] and Sierra [65]. Our simulations construct 3-level fat-tree networks using 72 radix switches.

Routing: Network-specific adaptive routings have used for the results shown in this paper for all the topologies because they provided better performance than static routing in all our experiments.

IV. SINGLE JOB SIMULATIONS

In single job simulations, all nodes of the prototype system are assigned to the same proxy application. Depending on the application, either four or sixteen *endpoints* are executed on each node. Here, endpoint refers to a control flow that generates communication, e.g. an MPI process. For each proxy application, we have attempted to find the best execution time by trying different routing schemes and task mappings. Since adaptive routing provides the best execution for all applications and systems, results for other routing schemes are omitted.

A. Communication-only Simulations

Fig. 3 presents the predicted execution time for the TRANSPOSE and FINE-GRAINED benchmarks executed in communication-only mode with 16 endpoints per node. In every iteration of TRANSPOSE, each endpoint exchanges a message with another endpoint that is far away from it in rank space. Hence, communication-only TRANSPOSE stresses the bisection bandwidth of the network. In the presented results, the execution time of the torus machine is taken as the base value that increases almost linearly with the message size.

Fig. 3 (a-b) show that the fat-tree machine consistently provides up to 40% faster execution in comparison to the torus machine for BALANCED INJECTION. This is expected since the bisection bandwidth of the fat-tree network is significantly greater than the bisection bandwidth of the torus network. Execution time on the dragonfly machine is significantly better than the torus machine for many message sizes, but is worse than the fat-tree machine though their networks have similar bisection bandwidths.

There is large variation in the performance of the dragonfly machine. Such variations are observed in the behavior of the dragonfly network in many results presented in the rest of this paper. These variations are most likely due to its hybrid UGAL-L routing scheme that makes greedy decisions based on limited local information [13]. This hypothesis is supported by the presence of links that are hot-spots for cases in which the performance of the dragonfly network is low. Nonetheless, performance obtained with hybrid UGAL-L routing is significantly better than the performance obtained with static routing and indirect routing.

For simulation of TRANSPOSE, the impact of using HIGH INJECTION instead of BALANCED INJECTION is marginal both in terms of absolute execution time and relative behavior among the different networks.

In the FINE-GRAINED benchmark, each endpoint exchanges small sized messages with approximately 1,000 partners.

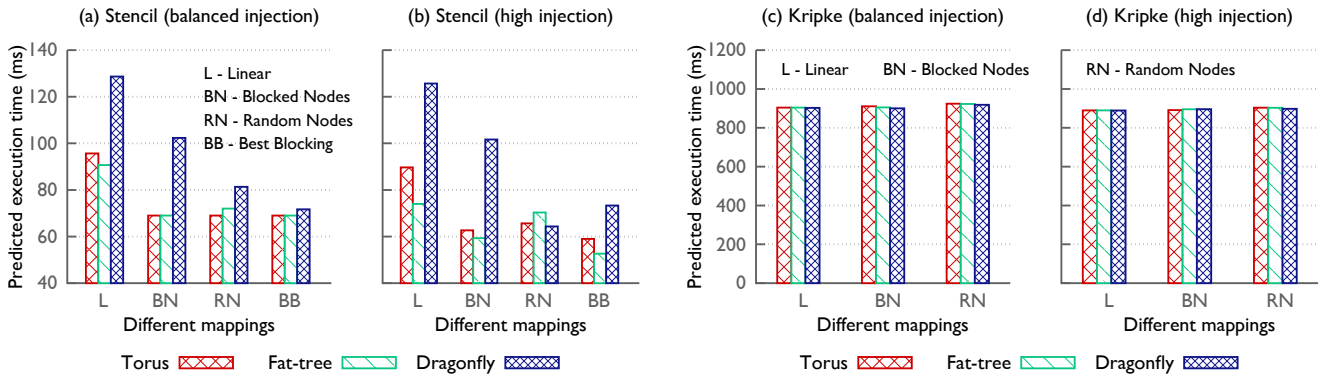

Fig. 4. **STENCIL**: Task mapping impacts performance on all machines. **KRIPKE**: Task mapping and network topology have a negligible effect on performance.

Fig. 3 (c) shows that when the message size is up to a few hundred bytes, the execution time does not change with the message size since the software delay, router delay, and link delay are the primary bottlenecks. For the smallest message size (100 bytes), the torus machine is fastest due to higher router and link counts, which helps in reducing the critical path overheads caused by the fixed delays. For simulation of FINE-GRAINED with all other message sizes, the benefit of higher router and link counts in a torus is normalized by contention for links caused by higher hop counts, and thus similar execution time is obtained for all the networks with BALANCED INJECTION

Increase in the bandwidth between NICs and routers (using HIGH INJECTION) has a positive effect on the dragonfly and torus machines, both of which provide better distribution of traffic over their links. This results in significant reduction in the execution time at relatively larger message sizes. In contrast, HIGH INJECTION does not affect either the traffic distribution or execution time on the fat-tree machine.

B. Grid-based Proxy Applications

In both STENCIL and KRIPKE, the 3D application domain is divided among endpoints arranged in a 3D Cartesian grid. In every iteration of the STENCIL proxy application, each endpoint exchanges boundary data with its six immediate neighbors and then performs relaxation-update. Fig. 4 (a-b) show results for executing STENCIL with 16 endpoints per node; each endpoint contains $180 \times 180 \times 180$ grid points with 40 variables [42]. The projected computation time used in these simulations for relaxation-update is 36 ms per iteration.

Task mapping has a significant impact on execution time of STENCIL on all systems. On torus and fat-tree machines with linear mapping, we found injection of data from NIC to be the primary bottleneck for STENCIL. Hence as better grouping of endpoints is performed, which reduces the amount of data that needs to be injected onto the network, the execution time reduces. On the dragonfly machine, grouping also helps reduce load on a few links that are otherwise overloaded due to the near-neighbor communication pattern.

With HIGH INJECTION, execution time is reduced on all systems for STENCIL. However, the fat-tree machine makes the best use of the increased injection bandwidth by grouping endpoints at the level of router and using the higher bandwidth for a large fraction of its communication.

In contrast to STENCIL in which all endpoints communicate simultaneously, KRIPKE performs a sweep that starts from corners of the particle grid and travels towards the center of the grid. Computation is performed both at the beginning of the iteration and when the sweep wavefront reaches a specific endpoint. These computation times are projected to be 52 ms and 1.1 ms, respectively, assuming 4 endpoints per node. Depending on their location in the grid, different endpoints send 1 to 256 messages of size 74 KB each.

Fig. 4 (c-d) show that mapping, network topology, and injection bandwidth have no impact on the execution time of KRIPKE despite high variation in link traffic within a network and among networks. This is because KRIPKE's sweep creates a linear dependence in the communication flow and hence all networks are under-utilized. For example, on the most loaded link in the torus network, only 70 MB data is transferred, though the link can transfer 11 GB data in the given time when fully utilized.

C. Proxy Applications with Collectives

We now present results on proxy applications with collective operations: FFT and QBOX-MINI. In FFT, endpoints are arranged as a 3D grid ($96 \times 48 \times 40$) and a 3D data grid of size either 2048^3 or 8192^3 is divided among them. One step of FFT computation is simulated by performing all-to-all operations within sub-communicators defined along the third dimension of the 3D grid of endpoints. In order to study the effect of communication, we do not allow grouping of endpoints within a sub-communicator onto a node. Computation time for FFT is projected to be 1.1 ms and 6.7 ms for the small and large grid, respectively.

Fig. 5 (a-b) show that similar performance is obtained for all systems with BALANCED INJECTION. A look at the per link traffic data reveals that despite up to $2\times$ difference in average traffic per link across different network topologies, links that form the bottleneck in different networks have similar number of bytes flowing through them. With HIGH INJECTION, the distribution of traffic improves for dragonfly

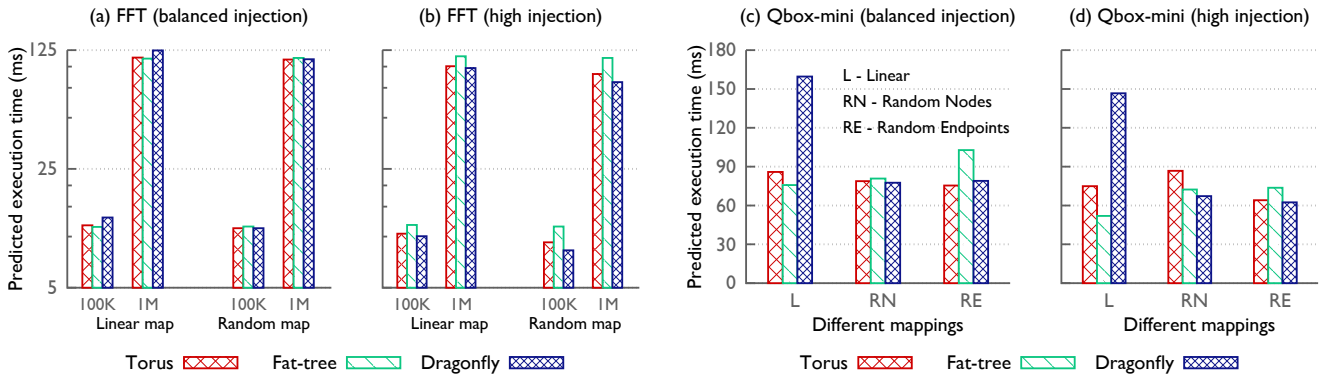

Fig. 5. **FFT**: Similar execution times are observed on all machines; HIGH INJECTION improves performance on dragonfly and torus machines by up to 27%. **QBOX-MINI**: Random mapping is necessary to obtain the best performance on the dragonfly machine; HIGH INJECTION improves performance on all machines.

and torus networks. As a result, up to 27% reduction in execution time is observed. In contrast, as seen before, the impact of higher injection on the fat-tree network is negligible.

In QBOX-MINI, endpoints are arranged in a 2D grid with column-major ordering. Domain data such as electronic states are divided among columns of endpoints in the 2D grid. Within a column, a given electronic state is divided among the endpoints. Every iteration begins with a pair of line-FFT computations on each endpoint followed by all-to-alls within the columns and a line-FFT on the endpoints. Then, all-reduce operation is performed within rows of the 2D grid of endpoints, followed by more computation.

Fig. 5 (c-d) show that by choosing the right type of mapping, similar performance can be obtained for all systems with BALANCED INJECTION. While a randomized mapping is the best for the dragonfly machine, a linear mapping that places endpoints within a column close to each other is not good. Random node mapping strikes a good balance and provides reasonable performance for all systems. Increasing injection bandwidth improves performance for all systems. For the fat-tree machine, the improvement is significant only when linear mapping is used because majority of communication is between the NICs and switches.

V. CAPABILITY WORKLOAD SIMULATIONS

In the capability workload simulations, four jobs are executed simultaneously on a given system. We fix the size of each of these jobs to be 11,520 nodes, i.e. one-fourth of the full system. Depending on the application, different message sizes, projected computation time, and number of endpoints per node are chosen. Also, based on the expected time per iteration, different applications are simulated for varying number of iterations to make them execute for similar amount of total time in the ideal scenario. Both TRANSPOSE and FINE-GRAINED proxy applications are also simulated with computation, unlike Section IV-A in which they were simulated in communication-only mode. Two types of job placements are simulated: linear distribution and random distribution of nodes among jobs. Best possible task mapping is used within each job based on the results from the previous section.

Fig. 6 presents the predicted execution time for each application simulated as part of six different workloads. Results from simulation of additional workloads are omitted due to lack of space. In Fig. 6 (a-b), predicted times for running four STENCIL and four TRANSPOSE jobs are shown. With linear job placement, all four jobs within a workload finish execution in similar time. This shows that all networks are able to minimize interference for symmetric workloads. However, when random job placement is used for TRANSPOSE, significantly different execution times are observed for the jobs running on the dragonfly machine. More experiments with minor changes in application configuration show that jobs running on the dragonfly network experience significant variations with random job placement.

In terms of absolute time, torus and fat-tree machines provide similar execution time for both workloads with BALANCED INJECTION. Dragonfly machine is only 7% slower for the 4-STENCILS workload, but is 53% slower for the 4-TRANSPOSES workload. This is because while the average traffic on links of the dragonfly network is similar to that of torus and fat-tree networks, the bottleneck links on the dragonfly network have up to 2× higher load. With randomized placement, load on the bottleneck links of torus either remains similar (for STENCIL) or decreases (for TRANSPOSE). For the dragonfly network, with randomized job placement, execution time and worst-case link traffic increases for STENCIL and decreases for TRANSPOSE.

Use of HIGH INJECTION improves performance in most cases in Fig. 6 (a-b), especially for the execution of four STENCIL jobs on the fat-tree machine. This trend is similar to the one observed in the previous section, where the best blocked task mapping limits a significant fraction of the communication to links connecting NICs and routers. For 4-TRANSPOSES workload, HIGH INJECTION also leads to better traffic distribution for all networks, thus reducing load on the most loaded links.

Fig. 6 (c-f) present prediction results for parallel workloads

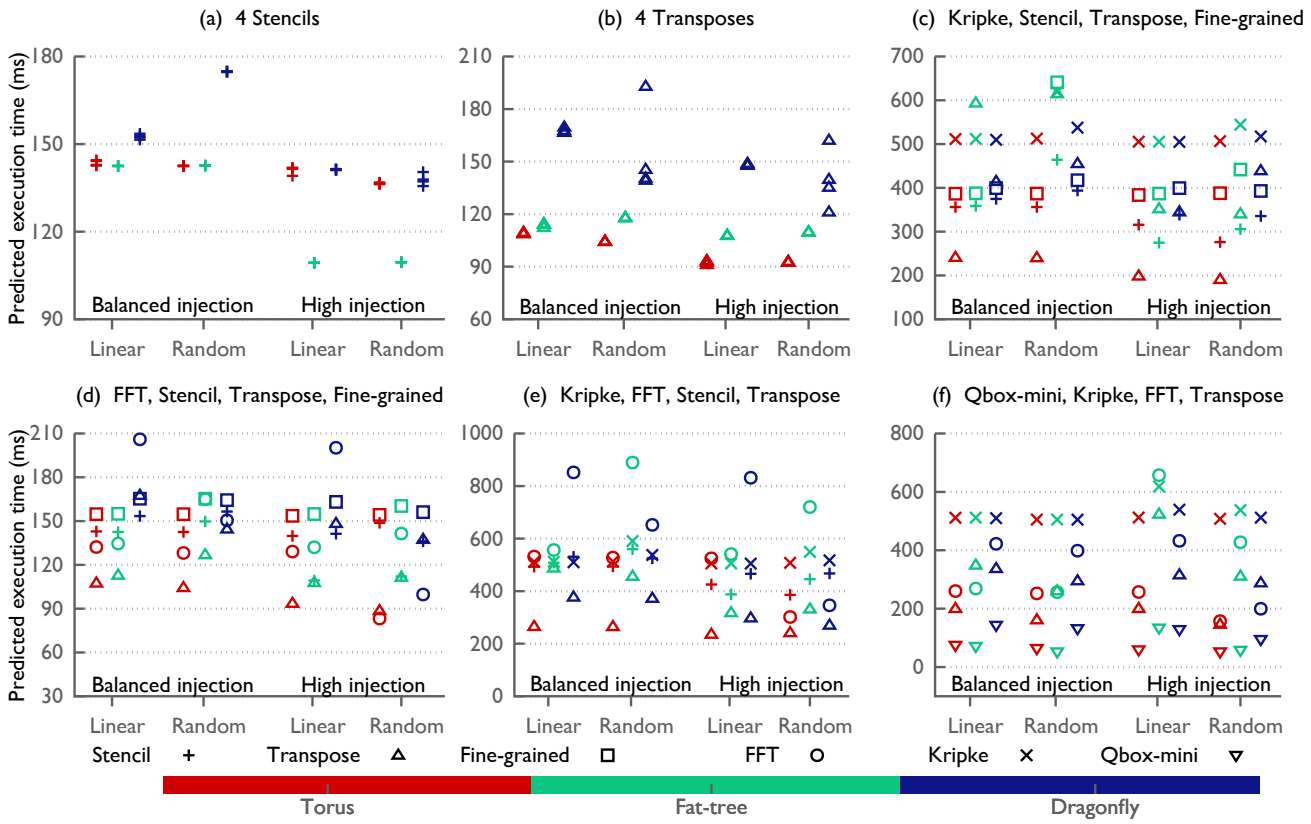

Fig. 6. Capability workloads with different job placements. The torus machine consistently provides good performance. Randomized placement can impact performance on the fat-tree machine. The dragonfly machine shows large variations with job placement and different injection bandwidths.

that comprise four different proxy applications each. In these results, the torus machine is consistently among the best performing systems. This is because it consistently provides a good distribution of traffic without excessively overloading a few links. Use of random job placement typically increases the average traffic for the torus network but does not increase the load on the bottleneck links significantly. Thus, it does not affect the execution time, except for TRANSPOSE which benefits from the higher available bisection bandwidth.

The performance of the fat-tree machine closely follows the torus machine but is typically worse than the torus machine, especially for TRANSPOSE, where the torus network is significantly faster. For all simulations, bytes on most congested links in the fat-tree network are higher than the corresponding value on the torus network. Random job placement leads to further increase in execution time on the fat-tree machines for many applications in the data sets.

The average execution time on the dragonfly machine is worst among the three networks, and so is the amount of traffic on the most congested links in the dragonfly network. In the cases where the linear job placement leads to bad performance, use of random job placement is able to better distribute the traffic and improve performance significantly for the dragonfly machine.

Use of HIGH INJECTION helps reduce the execution time for torus and dragonfly machines for four applications: STENCIL, TRANSPOSE, FFT, and QBOX-MINI. These results are consistent with the observations made in the analysis of single job simulations. For the fat-tree machine, HIGH INJECTION has a positive impact consistently only for STENCIL and QBOX-MINI. However, for FFT, it hurts performance significantly as shown in Fig. 6 (f). A detailed look at the link traffic statistics reveals that significant link load imbalance is observed in this simulation, and hence performance of all applications is negatively impacted.

VI. CAPACITY WORKLOAD SIMULATIONS

A common usage scenario for many large-scale systems called execution in *capacity* mode involves scheduling hundreds of jobs of varying sizes. In this section, we mimic execution in capacity mode by simulating 350 jobs of different sizes ranging from 32 nodes to 1,024 nodes with either 4 or 16 endpoints per node. For each job, one of the six proxy applications is selected in a uniformly random manner to determine the computation and control flow of the job.

Depending on the size of the job and the proxy application assigned to it, different message sizes and computation times are used in the simulations. These parameters are chosen such that each proxy application is either weak-scaled or strong-scaled. For a given combination of proxy application and job

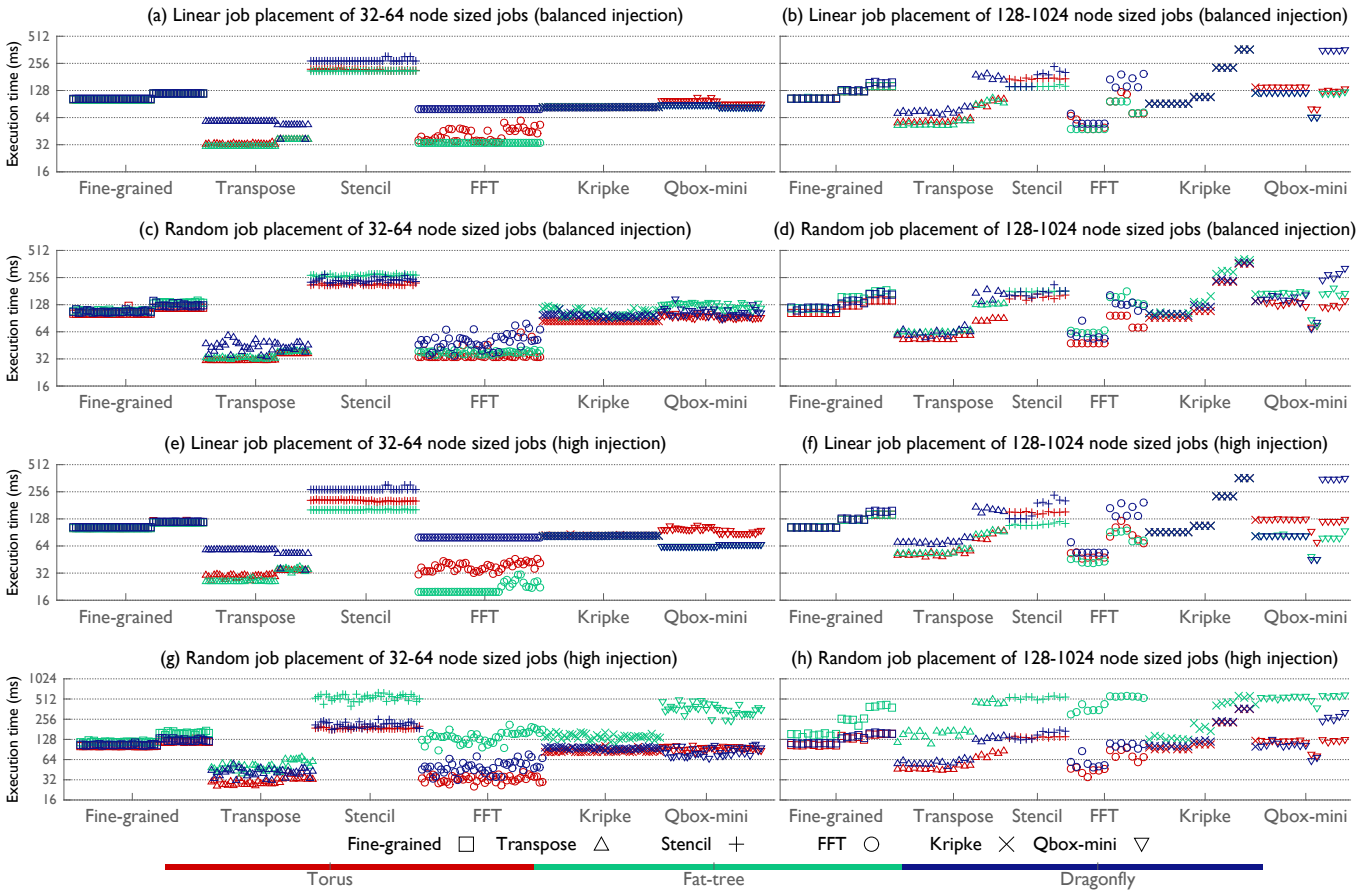

Fig. 7. Evaluating performance of a capacity workload with 350 jobs of sizes 32–1,024 nodes. Fat-tree and torus machines provide the best performance with linear and random job placement, respectively. With HIGH INJECTION, different network topologies provide the best performance for different applications.

size, the same trace is used to drive every job that is assigned that combination.

The capacity workload is simulated on all three prototype systems using either linear or random job placement and with BALANCED INJECTION or HIGH INJECTION. Fig. 7 presents the results obtained for these simulations wherein predicted execution time for each job is shown. The graphs on the left side show results for all 250 small jobs of size 32-64 nodes, while the right graphs are for the remaining 100 jobs of size 128-1,024 nodes. For ease of presentation, the jobs are arranged based on the proxy applications assigned to them. The jobs of each proxy application are sorted in an ascending order based on the number of nodes assigned to them.

With linear job placement using BALANCED INJECTION (Fig. 7 (a-b)), we observe that minimal variation and similar execution time is predicted for all jobs executing FINE-GRAINED and KRIPKE. For TRANSPOSE, STENCIL and FFT, the fat-tree machine provides the best performance.

The performance of the torus machine is similar to the fat-tree machine for TRANSPOSE, but the execution time on the dragonfly machine can be twice as high. For small jobs of STENCIL, the dragonfly machine's performance is worse than the fat-tree while showing significant variations for larger jobs. For QBOX-MINI, the execution time is minimum for the dragonfly machine except when the job size is 1,024 nodes. The torus machine does worse than both other systems in most cases for QBOX-MINI. Overall, the fat-tree machine provides the best performance which is also reflected in its traffic distribution. The load on the most congested link on the fat-tree network is 10% and 57% less than the corresponding values on the torus and dragonfly networks, respectively.

With random job placement using BALANCED INJECTION (Fig. 7 (c-d)), a few key differences are observed. For TRANSPOSE, the performance on the dragonfly machine improves and is closer to other machines, especially for large job-sizes. However, there is high variability across different TRANSPOSE jobs executed on the dragonfly machine. The execution time for STENCIL jobs running on the dragonfly machine is also reduced. In contrast, many TRANSPOSE and STENCIL jobs running on the fat-tree machine are now slower. Marginal impact is observed on runtime of similar jobs running on the torus machine.

With random job placement, performance of FFT on both torus and dragonfly machines improves significantly in comparison to linear job placement, while similar jobs executing on the fat-tree machine run longer. Higher variability and execution time is predicted for KRIPKE and QBOX-MINI on dragonfly and fat-tree machines but the performance of the

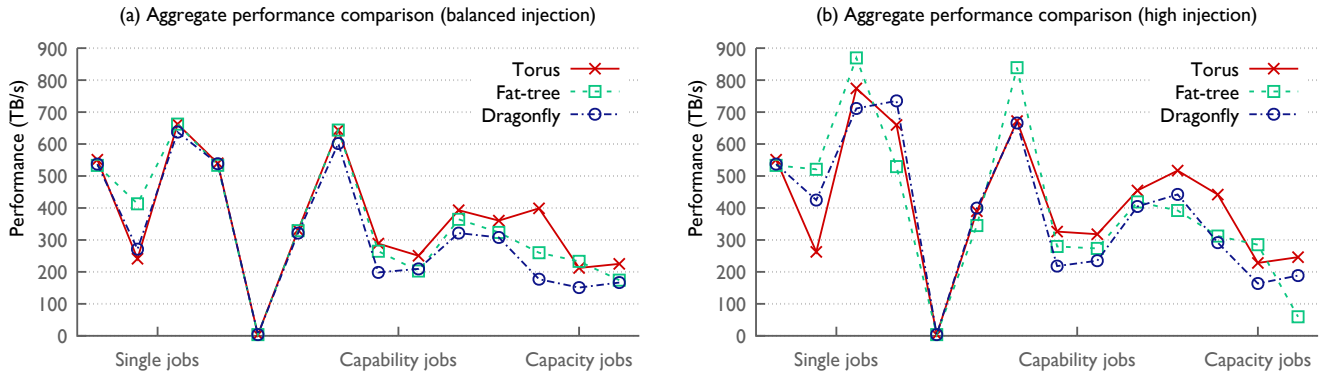

Fig. 8. Different networks provide the best performance for different workloads and configurations. Higher injection is beneficial for performance of dragonfly and fat-tree machines. For capacity jobs, results with both linear and random job placements are shown.

torus network is unaffected. In general, the torus network is predicted to perform the best with random job placement, with worst case link load being 30% and 50% less than the fat-tree and dragonfly networks respectively.

When HIGH INJECTION is used with linear job placement (Fig. 7 (e-f)), execution time for most applications (TRANSPOSE, FFT, STENCIL, and QBOX-MINI) on the fat-tree machine reduces. However, its impact on performance on torus and dragonfly machines is minimal, except for QBOX-MINI. The predicted time for QBOX-MINI on the dragonfly machine reduces significantly with HIGH INJECTION.

In contrast, when the injection bandwidth is increased for random job placement (Fig. 7 (g-h)), the performance on the fat-tree machine drops heavily although the distribution of traffic on its links does not change significantly. We were unable to find the root cause of this extreme slowdown on the fat-tree machine. The impact of HIGH INJECTION on torus and dragonfly networks is limited when using random job placement as was the case with linear job placement.

VII. AGGREGATING PERFORMANCE METRICS

We define the aggregated performance of a workload W that consists of n jobs as the sum of the effective bandwidths achieved by each individual job, i.e.,

$$P(W) = \sum_{i=1}^{n} \frac{comm_i}{t_i}$$

where $comm_i$ is the total number of bytes communicated by job i and t_i is the predicted execution time of job i.

Fig. 8 compares the three networks using the aggregated performance of all single jobs and all workloads. The x-axis shows from left to right: all single jobs (in the same order as Fig. 7), all capability workloads (in the same order as Fig. 6) and the capacity workload from Section VI with two placements (linear, random). In these plots, for each workload and network combination, the best aggregated performance from among the different job placements is shown.

For scenarios with BALANCED INJECTION (Fig. 8 (a)), we observe that all machines provide similar aggregated performance for most single jobs. This is because different task mappings are able to optimize network utilization, except for communication-only TRANSPOSE whose performance is limited by bisection bandwidth. For capability workloads simulated in this paper, the torus machine consistently achieves the best performance, while the fat-tree machine is marginally worse. The trend is reversed for the capacity workload, in which fat-tree achieves the best performance using linear job placement and torus is marginally worse (4%) using random job placement.

With HIGH INJECTION (Fig. 8 (b)), different machines provide the best aggregated performance for single jobs. For capability workloads, the aggregated performance increases on all networks in comparison to the BALANCED INJECTION scenario. However, the torus machine continues to achieve the highest aggregated performance for most capability workloads. Similarly, the fat-tree machine still has the highest aggregated performance for the capacity workload, which is 16% and 34% greater than the torus and dragonfly machines, respectively.

VIII. DISCUSSION AND FUTURE WORK

In order to utilize HPC systems effectively, new methods are needed to analyze diverse realistic workloads over an array of network alternatives. In this paper, we have demonstrated that TraceR built on top of CODES is capable of simulating large-scale systems executing such realistic and complex HPC workloads.

We have leveraged TraceR to demonstrate a comprehensive methodology useful for comparing performance of different network topologies. Using this methodology, specific prototype systems based on popular network topologies were explored for a variety of realistic workloads. For single job executions, we found that the use of task mapping leads to similar performance for all networks. With higher injection bandwidth, different networks achieve best performance. For multi-job workloads with a few large jobs, the torus network consistently achieves the best performance with the fat-tree network's performance being slightly worse. For the capacity workload designed by us, the aggregated performance of the fat-tree machine is the best. We also found that the

dragonfly machine is more likely to show higher variability in performance when multiple jobs are executed on it using randomized job placement.

Given the numerous parameters related to network design, and the diversity in workloads routinely run at different HPC centers, it is desirable to conduct similar detailed studies to find the optimal network for a given scenario. We hope that the new simulation capability and methodology presented in this paper can be deployed to make optimal choices based on the requirements of different supercomputing centers.

In the future, we plan to extend this methodology to include dollar cost estimates for network procurement and operation. By combining performance metrics with cost estimates, we can compare performance efficacy of different networks with respect to their procurement and operational costs. We also plan to simulate fat-tree network models with programmable connectivity and different routing schemes.

ACKNOWLEDGMENT

The authors thank Misbah Mubarak and John Jenkins for their help in this project. The authors also thank Adam Moody for sharing information on fat-tree networks and Chad Wood for proofreading the paper.

This work was performed under the auspices of the U.S. Department of Energy (DOE) by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-690662). This research was partly funded by the following grants: NSF SI2-SSI Grant ID ACI 13-39715, U.S. DOE Award Number DE-NA0002374 and the Blue Waters sustained-petascale computing project (NSF Award Number OCI 07-25070). This research used resources at the Argonne Leadership Computing Facility and Oak Ridge Leadership Computing Facility, both supported by the Office of Science of the U.S. DOE.

REFERENCES

[1] M. Ould-Khaoua and H. Sarbazi-Azad, "An analytical model of adaptive wormhole routing in hypercubes in the presence of hot spot traffic," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 3, pp. 283–292, 2001.

[2] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "Logp: Towards a realistic model of parallel computation," in *Fourth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming PPOPP*, San Diego, CA, May 1993.

[3] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "Loggp: incorporating long messages into the logp modelone step closer towards a realistic model for parallel computation," in *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, ser. SPAA '95. New York, NY, USA: ACM, 1995, pp. 95–105. [Online]. Available: http://doi.acm.org/10.1145/215399.215427

[4] D. Martinez, J. Cabaleiro, T. Pena, F. Rivera, and V. Blanco, "Accurate analytical performance model of communications in mpi applications," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, May 2009, pp. 1–8.

[5] M. J. Clement and M. J. Quinn, "Analytical performance prediction on multicomputers," in *Supercomputing*, 1993, pp. 886–894.

[6] C. A. Moritz and M. I. Frank, "Logpc: Modeling network contention in message-passing programs," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 1, pp. 254–263, Jun. 1998.

[7] K. L. Spafford and J. S. Vetter, "Aspen: A domain specific language for performance modeling," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 84:1–84:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389110

[8] 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*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 336–347. [Online]. Available: http://dx.doi.org/10.1109/SC.2014.33

[9] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, J. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *IEEE International Symposium on Performance Analysis of Systems and Software*, 2013.

[10] G. Zheng, T. Wilmarth, P. Jagadishprasad, and L. V. Kalé, "Simulation-based performance prediction for large parallel machines," in *International Journal of Parallel Programming*, vol. 33, no. 2-3, 2005, pp. 183–207.

[11] T. Hoefler, T. Schneider, and A. Lumsdaine, "LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model," in *Proceedings of the 19th ACM International Symposium on HPDC*. ACM, Jun. 2010, pp. 597–604.

[12] C. L. Janssen, H. Adalsteinsson, S. Cranford, J. P. Kenny, A. Pinar, D. A. Evensky, and J. Mayo, "A simulator for large-scale parallel computer architectures." *IJDST*, vol. 1, no. 2, pp. 57–73, 2010.

[13] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," *SIGARCH Comput. Archit. News*, vol. 36, pp. 77–88, June 2008.

[14] J. Kim, W. J. Dally, and D. Abts, "Flattened butterfly: A cost-efficient topology for high-radix networks," *SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 126–137, Jun. 2007. [Online]. Available: http://doi.acm.org/10.1145/1273440.1250679

[15] 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*, ser. SC '14. IEEE Press, 2014, pp. 348–359.

[16] G. Kathareios, C. Minkenberg, B. Prisacari, G. Rodriguez, and T. Hoefler, "Cost-Effective Diameter-Two Topologies: Analysis and Evaluation." ACM, Nov. 2015, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC15).

[17] N. Jiang, L. Dennison, and W. J. Dally, "Network endpoint congestion control for fine-grained communication," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: ACM, 2015. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807600

[18] 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.

[19] E. Mikida, N. Jain, E. Gonsiorowski, P. D. Barnes, Jr., D. Jefferson, C. D. Carothers, and L. V. Kale, "Towards pdes in a message-driven paradigm: A preliminary case study using charm++," in *ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS)*, ser. SIGSIM PADS '16 (to appear). ACM, May 2016.

[20] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns, "Enabling parallel simulation of large-scale hpc network systems," *IEEE Trans. Parallel Distrib. Syst.*, 2016.

[21] M. Mubarak, C. D. Carothers, R. Ross, and P. Carns, "A case study in using massively parallel simulation for extreme-scale torus network codesign," in *SIGSIM PADS'14*, 2014, pp. 27–38.

[22] ——, "Modeling a million-node dragonfly network using massively parallel discrete-event simulation," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, Nov 2012, pp. 366–376.

[23] B. Acun, N. Jain, A. Bhatele, M. Mubarak, C. D. Carothers, and L. V. Kale, "Preliminary evaluation of a parallel trace replay tool for hpc network simulations," in *Workshop on Parallel and Distributed Agent-Based Simulations*, ser. PADABS, EURO-PAR, Aug. 2015.

[24] "MPI: A Message Passing Interface Standard," in *MPI Forum*, http://www.mpi-forum.org/.

[25] B. Acun, A. Gupta, N. Jain, A. Langer, H. Menon, E. Mikida, X. Ni, M. Robson, Y. Sun, E. Totoni, L. Wesolowski, and L. Kale, "Parallel Programming with Migratable Objects: Charm++ in Practice," ser. SC, 2014.

[26] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving performance via mini-applications," Sandia National Laboratories, Tech. Rep., September 2009.

[27] "CORAL Collaboration Benchmark Codes," https://asc.llnl.gov/CORAL-benchmarks.

[28] "NERSC-8 / Trinity Benchmarks." [Online]. Available: http://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks

[29] M.Blumrich, D.Chen, P.Coteus, A.Gara, M.Giampapa, P.Heidelberger, S.Singh, B.Steinmacher-Burow, T.Takken, and P.Vranas, "Design and Analysis of the Blue Gene/L Torus Interconnection Network," *IBM Research Report*, December 2003.

[30] V. Puente, R. Beivide, J. Gregorio, J. Prellezo, J. Duato, and C. Izu, "Adaptive bubble router: a design to improve performance in torus networks," in *Parallel Processing, 1999. Proceedings. 1999 International Conference on*, 1999, pp. 58–67.

[31] L. Schwiebert and D. N. Jayasimha, "On measuring the performance of adaptive wormhole routing," *hipc*, vol. 00, p. 336, 1997.

[32] S. Kumar, Y. Sabharwal, R. Garg, and P. Heidelberger, "Optimization of all-to-all communication on the blue gene/l supercomputer," in *Proceedings of the 2008 37th International Conference on Parallel Processing*, ser. ICPP '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 320–329. [Online]. Available: http://dx.doi.org/10.1109/ICPP.2008.83

[33] D. Chen, N. Eisley, P. Heidelberger, S. Kumar, A. Mamidala, F. Petrini, R. Senger, Y. Sugawara, R. Walkup, B. Steinmacher-Burow, A. Choudhury, Y. Sabharwal, S. Singhal, and J. J. Parker, "Looking under the hood of the ibm blue gene/q network," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 69:1–69:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389090

[34] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1402946.1402967

[35] X. Yuan, "On nonblocking folded-clos networks in computer communication environments," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, May 2011, pp. 188–196.

[36] K. Underwood, M. Levenhagen, and A. Rodrigues, "Simulating red storm: Challenges and successes in building a system simulation," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1 –10.

[37] E. Zahavi, G. Johnson, D. J. Kerbyson, and M. Lang, "Optimized infinibandtm fat-tree routing for shift all-to-all communication patterns," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 2, pp. 217–231, 2010. [Online]. Available: http://dx.doi.org/10.1002/cpe.1527

[38] B. Prisacari, G. Rodriguez, C. Minkenberg, and T. Hoefler, "Fast pattern-specific routing for fat tree networks," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 4, pp. 36:1–36:25, Dec. 2013. [Online]. Available: http://doi.acm.org/10.1145/2555289.2555293

[39] A. Bhatele, N. Jain, W. D. Gropp, and L. V. Kale, "Avoiding hot-spots on two-level direct networks," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 76:1–76:11.

[40] A. Bhatele, "Automating Topology Aware Mapping for Supercomputers," Ph.D. dissertation, Dept. of Computer Science, University of Illinois, August 2010, http://hdl.handle.net/2142/16578.

[41] T. Hoefler and M. Snir, "Generic topology mapping strategies for large-scale parallel architectures," in *Proceedings of the international conference on Supercomputing*, ser. ICS '11. New York, NY, USA: ACM, 2011, pp. 75–84.

[42] "MiniGhost finite difference mini-application." http://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks/minighost/.

[43] M. Collaboration, "MIMD Lattice Computation (MILC) Collaboration Home Page," http://www.physics.indiana.edu/~sg/milc.html.

[44] "The weather research & forecasting model website," http://wrf-model.org.

[45] A. Gupta and V. Kumar, "Scalability of parallel algorithms for matrix multiplication," *Parallel Processing, 1993. ICPP 1993. International Conference on*, vol. 3, pp. 115–123, Aug. 1993.

[46] N. Jain, E. Bohm, E. Mikida, S. Mandal, M. Kim, P. Jindal, Q. Li, S. Ismail-Beigi, G. Martyna, and L. Kale, "Openatom: Scalable ab-initio molecular dynamics with diverse capabilities," in *International Supercomputing Conference*, ser. ISC HPC '16 (to appear), 2016.

[47] M. Eleftheriou, B. G. Fitch, A. Rayshubskiy, T. J. C. Ward, and R. S. Germain, "Scalable framework for 3D FFTs on the Blue Gene/L supercomputer: Implementation and early performance measurements," *IBM Journal of Research and Development*, vol. 49, no. 2/3, 2005.

[48] C. H. Still, R. L. Berger, A. B. Langdon, D. E. Hinkel, L. J. Suter, and E. A. Williams, "Filamentation and forward brillouin scatter of entire smoothed and aberrated laser beams," *Physics of Plasmas*, vol. 7, no. 5, p. 2023, 2000.

[49] J. Carrier, L. Greengard, and V. Rokhlin, "A fast adaptive multipole algorithm for particle simulations," *SIAM J. Sci. Stat. Comput.*, vol. 9, Jul. 1988.

[50] J.-S. Yeom, A. Bhatele, K. R. Bisset, E. Bohm, A. Gupta, L. V. Kale, M. Marathe, D. S. Nikolopoulos, M. Schulz, and L. Wesolowski, "Overcoming the scalability challenges of epidemic simulations on blue waters," in *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*, ser. IPDPS '14. IEEE Computer Society, May 2014.

[51] A. J. Kunen, T. S. Bailey, and P. N. Brown, "KRIPKE - a massively parallel transport mini-app," Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Tech. Rep., 2015.

[52] F. Gygi, E. W. Draeger, M. Schulz, B. R. de Supinski, J. A. Gunnels, V. Austel, J. C. Sexton, F. Franchetti, S. Kral, C. W. Ueberhuber, and J. Lorenz, "Large-scale electronic structure calculations of high-z metals on the bluegene/l platform," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1188455.1188502

[53] Y. Ajima, S. Sumimoto, and T. Shimizu, "Tofu: A 6d mesh/torus interconnect for exascale computers," *Computer*, vol. 42, pp. 36–40, 2009.

[54] Cray Inc., "Cray XE6 Specifications," http://www.cray.com/Assets/PDF/products/xe/CrayXE6Brochure.pdf, 2010.

[55] S. Kumar, A. Mamidala, D. Faraj, B. Smith, M. Blocksome, B. Cernohous, D. Miller, J. Parker, J. Ratterman, P. Heidelberger, D. Chen, and B. Steinmacher-Burow, "PAMI: A parallel active message interface for the BlueGene/Q supercomputer," in *Proceedings of 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Shanghai, China, May 2012.

[56] *Cray T3D System Architecture Overview*, Cray Research, Inc., March 1993.

[57] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li, N. Ni, and R. Rajamony, "The PERCS High-Performance Interconnect," in *2010 IEEE 18th Annual Symposium on High Performance Interconnects (HOTI)*, August 2010, pp. 75–82.

[58] G. Faanes, A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard, "Cray cascade: A scalable hpc system based on a dragonfly network," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, Nov 2012.

[59] "AURORA, Argonne National Laboratory," http://aurora.alcf.anl.gov/.

[60] 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 *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, ser. HPDC '14. ACM, 2014, pp. 129–140. [Online]. Available: http://doi.acm.org/10.1145/2600212.2600225

[61] C. Leiserson, "Fat-trees: Universal Networks for Hardware-Efficient Supercomputing," *IEEE Transactions on Computers*, vol. 34, no. 10, October 1985.

[62] "Lonestar supercomputer at TACC," https://www.tacc.utexas.edu/systems/lonestar.

[63] "Stampede supercomputer at TACC," https://www.tacc.utexas.edu/stampede/.

[64] "SUMMIT, Oak Ridge National Laboratory," https://www.olcf.ornl.gov/summit/.

[65] "Sierra, Lawrence Livermore National Laboratory," https://asc.llnl.gov/coral-info.