Predicting the Performance Impact of Different Fat-Tree Configurations

Nikhil Jain
Lawrence Livermore National Laboratory
nikhil@llnl.gov

Abhinav Bhatele
Lawrence Livermore National Laboratory
bhatele@llnl.gov

Louis H. Howell
Lawrence Livermore National Laboratory
howell4@llnl.gov

David Böhme
Lawrence Livermore National Laboratory
boehme3@llnl.gov

Ian Karlin
Lawrence Livermore National Laboratory
karlin1@llnl.gov

Edgar A. León
Lawrence Livermore National Laboratory
leon@llnl.gov

Misbah Mubarak
Argonne National Laboratory
mmubarak@anl.gov

Noah Wolfe
Rensselaer Polytechnic Institute
wolfen@rpi.edu

Todd Gamblin
Lawrence Livermore National Laboratory
tgamblin@llnl.gov

Matthew L. Leininger
Lawrence Livermore National Laboratory
leininger4@llnl.gov

ABSTRACT

The fat-tree topology is one of the most commonly used network topologies in HPC systems. Vendors support several options that can be configured when deploying fat-tree networks on production systems, such as link bandwidth, number of rails, number of planes, and tapering. This paper showcases the use of simulations to compare the impact of these design options on representative production HPC applications, libraries, and multi-job workloads. We present advances in the TraceR-CODES simulation framework that enable this analysis and evaluate its prediction accuracy against experiments on a production fat-tree network. In order to understand the impact of different network configurations on various anticipated scenarios, we study workloads with different communication patterns, computation-to-communication ratios, and scaling characteristics. Using multi-job workloads, we also study the impact of inter-job interference on performance and compare the cost-performance tradeoffs.

CCS CONCEPTS

• **Networks → Network performance modeling**; **Network simulations**; **Network performance analysis**;

KEYWORDS

fat-tree topology, network simulation, performance prediction, procurement

ACM Reference Format:
Nikhil Jain, Abhinav Bhatele, Louis H. Howell, David Böhme, Ian Karlin, Edgar A. León, Misbah Mubarak, Noah Wolfe, Todd Gamblin, and Matthew L. Leininger. 2017. Predicting the Performance Impact of Different Fat-Tree Configurations. In *Proceedings of SC17*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3126908.3126967

1 INTRODUCTION

The fat-tree topology [31] is expected to be used for building the interconnection networks of many next-generation high performance computing (HPC) systems, e.g. Sierra at Lawrence Livermore National Laboratory (LLNL) [8] and Summit at Oak Ridge National Laboratory [10]. It is currently used in many production systems, ranging from small clusters with a few hundred nodes to multi-petaFlop/s supercomputers [9]. Such extensive use of fat-tree networks is partly driven by their easily extensible construction from off-the-shelf commodity hardware. The fat-tree topology also provides high bisection bandwidth and a relatively low diameter among the available alternatives for a given node count.

The capabilities of networks are typically improved commensurately with the computational power of HPC systems to prevent networks from negatively impacting the performance of applications. For fat-tree networks, this is currently being done by increasing the bandwidth of links and by using multi-rail and multi-plane networks [20]. However, network improvements increase procurement and operational costs.

Since the total budget available for procurement and operation of supercomputers is typically fixed, HPC centers strive to strike a

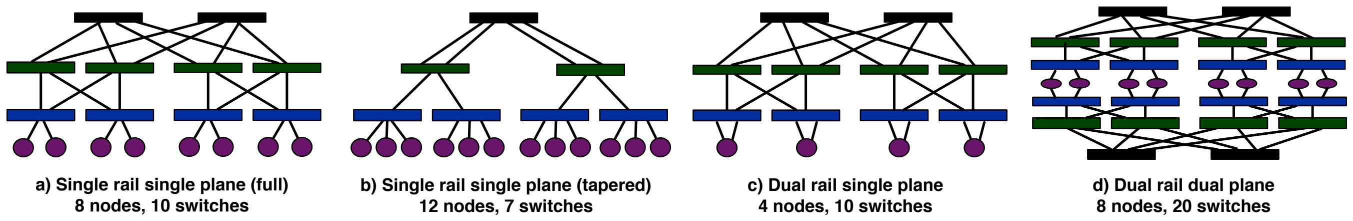

Figure 1: Examples of design options for fat-tree networks.

balance between the computational and communication capabilities of their systems. The increasing cost of networks presents a challenge in this regard. On one hand, HPC centers want to minimize the networking cost so that more computational resources can be purchased and made available for the users. On the other hand, reduced network capabilities can slow down some applications and offset the gains provided by additional computational resources. Thus, HPC centers are interested in understanding the impact of reduced network capabilities on performance and mitigating the risks of reducing those capabilities.

As a result, it is imperative to develop prediction methodologies that can assist in evaluating the performance of applications and multi-job workloads on available network alternatives. For typical HPC data centers, a significant fraction of system time is spent in running a few production applications (each using 5–30% of total nodes in the system [32]). Thus, in this paper, we focus on predicting the impact of alternative fat-tree configurations on a set of such representative applications and libraries.

Several runtime characteristics are expected to have a significant effect on the performance of current applications on next-generation systems. We focus on predicting the impact of: 1) computation scaling of an application on its performance, 2) the communication pattern of an application on its scaling, and 3) the impact of inter-job interference on performance.

A robust, reliable simulation framework can play an important role in performing studies that can answer these questions. The need for simulations arises because existing systems offer limited options for studying future networks. This is because future networks will be inherently different from current networks. We also cannot change the hardware and/or change the parameters of production networks without disrupting existing production workloads.

The simulation framework used for prediction studies should be able to simulate production applications and libraries running on thousands of MPI processes. Use of production codes is critical because complex communication characteristics of many production codes are hard to capture in simplified proxy applications. However, accurately simulating production applications is challenging because the effects of millions of MPI communication calls and low-level transient contention need to be captured.

TraceR-CODES [27, 35] is a scalable simulation framework for predicting performance on HPC networks. In this paper, we extend the TraceR-CODES framework with multiple capabilities required to enable low-effort simulations of production applications and libraries on fat-tree alternatives. The capabilities added include support for OTF2 trace format [6], modeling MPI protocols and collectives, and fat-tree network options such as tapered, multi-rail, and multi-plane. We then use the extended framework to predict the performance impact of different network configurations on the following applications and libraries: Hypre [22], Atratus, Mercury [16], MILC [15], ParaDiS [13], pF3D [39], and Qbox [23].

The primary contributions of this work are:

- Advances in the TraceR-CODES framework that enable low-effort, accurate simulations of production applications.
- Validation of the TraceR-CODES framework for many parallel codes including production applications.
- Performance and interference predictions for production applications, libraries, and multi-job workloads on a range of potential future network designs.

We also assess the suitability of different fat-tree configurations for the applications and workloads simulated in this paper. Note that the choice of input problems can potentially alter the characteristics of many applications, and hence some of the results presented may not apply to inputs that significantly alter the behavior of an application.

2 FAT-TREE NETWORKS

The fat-tree topology is a tree-based topology in which bandwidth of edges increases near the top (root) of the tree [30]. Practical deployments of fat-tree in most supercomputers resemble the folded-Clos topology as shown in Figure 1(a). In this set up, many routers of same radix are grouped together to form *core* switches and provide high bandwidth. The fat-tree shown in Figure 1(a) is a *full* fat-tree: the total bandwidth within a level does not decrease as we move from nodes connected to the leaf switches toward higher levels.

In order to reduce the cost of the network, tapering can be deployed to connect more nodes per leaf switch (Figure 1(b)). This reduces the total bandwidth at higher levels but also lowers the number of switches and links required to connect the same number of nodes in comparison to the full fat-tree.

On the other hand, when higher bandwidth is desired, each node can be provided multiple ports (rails) to inject traffic at a higher rate into the leaf switches. The multiple ports can either be connected to switches in the same plane as shown in Figure 1(c) or to disjoint planes as shown in Figure 1(d). In both cases, fewer nodes can be connected using the same quantity of network resources in comparison to the single rail fat-tree. Either of these configurations can also be tapered to retain high injection bandwidth at the nodes, but reduce cost by reducing the bandwidth at the higher levels.

Table 1: Fat-tree configurations currently available.

Configuration	Link b/w	#rails	#planes	Tapering
SR-EDR	100 Gb/s	1	1	1:1
DRP-T-EDR	100 Gb/s	2	2	2:1
DRP-EDR	100 Gb/s	2	2	1:1
SR-HDR	200 Gb/s	1	1	1:1
DR-T-HDR	200 Gb/s	2	1	2:1
DR-HDR	200 Gb/s	2	1	1:1

Currently, all of the above configurations are offered by multiple vendors including Mellanox and Intel. In addition, several options are available for bandwidth of individual links – FDR (56 Gb/s), EDR (100 Gb/s), HDR (200 Gb/s). The combinatorial choices offered by these options make the task of finding the most suitable configuration for HPC centers and applications difficult. We address this challenge by showing that simulations of applications on possible configurations (Table 1) can provide key insights and reliable data points critical to this decision-making process.

3 APPLICATION CHARACTERISTICS

In this section, we briefly describe the codes used in this study and analyze their communication characteristics. The motivation for this analysis is two-fold. First, it helps understand the performance trends observed for various codes on different network configurations. Second, it can be used to find generic trends in impact of network configurations based on specific results observed for different applications.

The codes used in this study include applications and libraries that are either run in production at HPC centers (Hypre, Mercury, MILC, ParaDiS, pF3D, Qbox), or represent codes run in production (Atratus). These codes span a wide range of physics and mathematical domains including Monte Carlo, first-principles molecular dynamics, transport, plasma interactions, structured and unstructured grids, and sparse linear algebra.

Hypre [22] is a parallel linear solvers library developed at LLNL and is used by many production applications. For this study, we use the Algebraic Multigrid (AMG) Solver on a 2D diffusion problem using structured Adaptive Mesh Refinement (AMR). We run a weak scaling problem so the number of mesh points is proportional to the number of MPI processes. The tests are set up within a larger application code but tracing is limited to the operations taking place during the Hypre setup and solve phases.

Atratus extends MULARD [5], a high order, finite element based multigroup radiation diffusion code that uses a 3D unstructured mesh, by including more advanced physics and discretizations. It is used primarily as a research tool to explore future programming paradigms with data flow and computations important to LLNL applications. Atratus uses MFEM [4], which invokes Hypre solvers but they are more specialized than the standard AMG solver we run for the Hypre tests.

Mercury [16] is a production Monte Carlo application developed at LLNL as a general-purpose particle transport code. For these experiments, we run a 3D neutron transport eigenvalue problem using realistic material properties and a load balancing scheme based on domain replication. The number of particles are weak-scaled with the number of MPI processes but the mesh geometry is held constant.

Table 2: Problem sizes run on 16K MPI processes.

App	Description of input problem
Hypre	Diffusion with tiled pattern, 34,320 unknowns per process
Atratus	CrookedPipe, 28,740 unknowns per process
Mercury	2500 particles per process
MILC	su_rmd, 16,384 grid points per process
ParaDiS	5.2 million nodes total
pF3D	3.9M grid points per process
Qbox	Gold benchmark, 512 atoms total

MILC [15] is a widely used parallel application suite for studying quantum chromodynamics (QCD), the theory of strong interactions of subatomic physics. We have used the `su3_rmd` application in which quark fields and MPI processes are defined as a 4D grid of space time points.

ParaDiS [13] is a large-scale dislocation dynamics simulation code used to study the fundamental mechanisms of plasticity at LLNL. ParaDiS couples direct N^2 calculation of forces between dislocation line segments with a fast multipole method (FMM) to capture remote interactions. It uses a hierarchical decomposition scheme where the simulation volume is recursively divided into slabs along each physical direction.

pF3D [39] is used for simulating laser-plasma interactions in experiments conducted at the National Ignition Facility at LLNL. It solves coupled wave equations for the laser light, the scattered light, and the plasma. pF3D uses a regular 3D Cartesian grid and spatial decomposition with 2D FFTs in the planes and beam propagation across the planes.

Qbox [23] is a first-principles molecular dynamics code that is used to compute the electronic structure of atoms, molecules, solids, and liquids within the Density Functional Theory (DFT) formalism. It won the Gordon Bell award at SC 2006 and is widely used for studying atomic properties of heavy metals. Qbox divides the atoms and their states among a 2D grid of MPI processes, and makes heavy use of FFTs and linear algebra libraries such as Scalapack and BLAS.

For each of the above codes, we worked with their developers and/or frequent users to construct representative scientific input problems for weak-scaled executions on 1K to 16K MPI processes. These problems were run either on an Infiniband-Xeon cluster (Quartz) [7] with 32 cores per node, or on an IBM Blue Gene/Q system (rzUSeq, Vulcan) with 16 cores per node. Table 2 lists the input problems that were run on 16K MPI processes. Using profiling data collected from runs on 16K processes, we now summarize the communication characteristics of these applications. The results presented here do not include MPI calls used in the initialization step of applications. As mentioned before, the choice of input problems

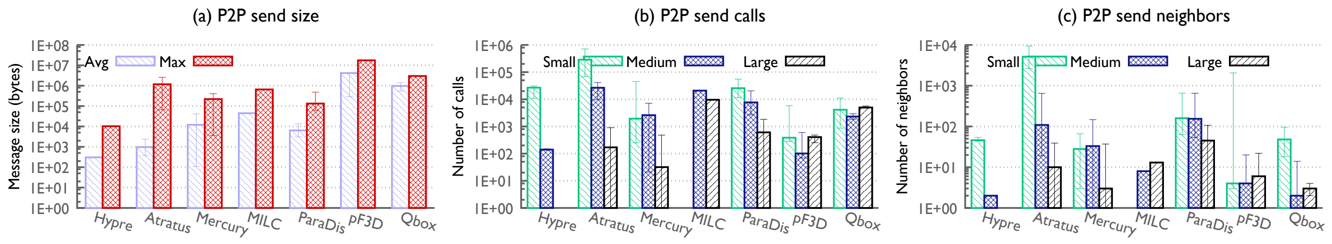

Figure 2: Communication characteristics for point-to-point operations across applications.

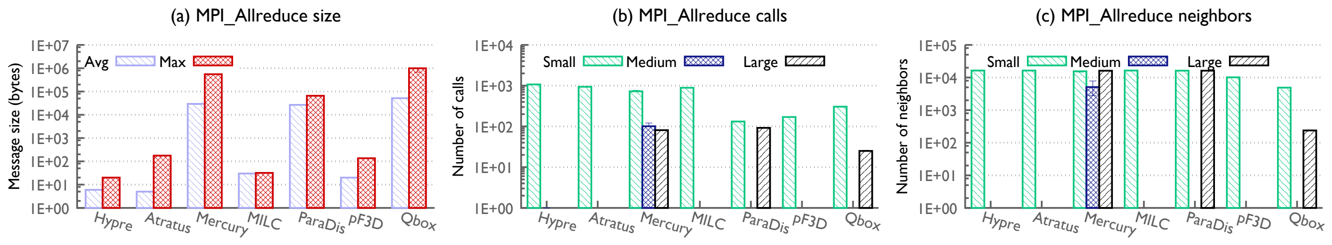

Figure 3: MPI_Allreduce characteristics across applications.

Table 3: Collective operations usage across applications. Each cell entry (a/b/c) represents three data points: a - message size, b - number of calls, and c - size of communicator.

Operation	Hypre and Atratus	Mercury	MILC	ParaDiS	pF3D	Qbox
Bcast	Small/Medium/Large	All/Large/Large	×	×	Small/Medium/Large	All/Large/Large
Reduce	×	All/Large/Large	×	×	×	All/Large/Medium
Alltoall	×	Small/Medium/Large	×	×	Large/Large/Small	Small/Large/Medium
Allgather	×	Small/Large/Large	×	Small/Small/Large	Small/Small/Medium	×

can potentially alter the characteristics of many of these applications, and hence some of the results presented may not apply for all problems run on the applications.

Point-to-point communication: Figure 2 presents the summary for point-to-point send operations performed by all the applications. Results for point-to-point receive operations are similar to the results for send operations, and thus not shown. Figure 2(a) shows the minimum, average, and maximum across processes of the average and maximum size of messages sent by individual processes. The maximum size for most processes across all applications except Hypre is greater than 100 KB. However, the average size is less than 1 KB for Hypre and Atratus, and greater than 1 MB for pF3D and Qbox.

In Figures 2(b,c), the distribution of message sends is shown based on their size. Messages with size less than 512 bytes are considered *small*, less than 64 KB are marked *medium*, and rest are termed *large*. These cutoffs points are chosen as they are currently the recommended values for switching from short to eager and eager to rendezvous protocols on Performance Scaled Messaging 2 (PSM2) [1]. For several codes (Hypre, Atratus, Mercury, and ParaDiS), a large fraction of message sends are either small or medium. The number of unique neighbors with which these messages are communicated is high, especially for small messages. In contrast, for MILC, pF3D, and Qbox, the number of large messages is high and the number of neighbors to which these messages are sent is small.

In summary, we can divide the applications into two sets: one with predominantly small/medium sized messages and a large number of neighbors (Hypre, Atratus, and Mercury), and another with many large message sends to a few neighbors (MILC, pF3D, and Qbox). ParaDis does not fall in either category since it has many small/medium sized messages and several large sized messages.

Collective communication: Figure 3 presents the summary for `MPI_Allreduce`, which is the most commonly used collective operation across applications. In these results, the criterion for classifying the operations based on their message size is the same as that for point-to-point messages. Figure 3 shows that almost all applications make hundreds of `MPI_Allreduce` calls of small size over all MPI processes. Mercury, ParaDiS, and Qbox also invoke many large sized `MPI_Allreduces`. Among them, Qbox's calls are over medium-sized communicators, while Mercury's and ParaDiS's calls are over large communicators.

Since most other collective calls are made by a subset of applications, they are succinctly presented in Table 3. Note that, for `MPI_Alltoall` and `MPI_Allgather`, the message size considered for marking calls as small, medium, or large is the amount of data

Table 4: Simplified qualitative summary of applications' communication characteristics.

App	P2P	Collectives
Hypre	light	light allreduce, bcast
Atratus	light	light allreduce, bcast
Mercury	light	light alltoall, allgather; heavy allreduce, bcast, reduce
MILC	heavy	light allreduce
ParaDiS	medium	light allgather; heavy allreduce
pF3D	heavy	light allreduce, bcast, allgather; heavy alltoall
Qbox	heavy	light alltoall; heavy allreduce, bcast, reduce

exchanged between a pair of MPI processes. In addition, if only tens of calls are made or if only tens of processes are part of a communicator, those calls are marked *small*. If the number of calls or communicator size is a few hundreds, we term the calls *medium*, and *large* otherwise.

Table 3 shows that Mercury and Qbox make heavy use of `MPI_Bcast` and `MPI_Reduce` over medium and large communicators for all categories of message sizes. In contrast, Hypre, Atratus, and pF3D invoke small sized broadcast operations. ParaDiS only invokes small sized `MPI_Allgather` a few times. `MPI_Allgather` is also used by Mercury and pF3D for small messages. pF3D also makes heavy use of `MPI_Alltoall` over small communicators, while Mercury and Qbox invoke `MPI_Alltoall` with small messages over medium to large communicators. Table 4 summarizes these findings.

4 SIMULATION FRAMEWORK

The TraceR-CODES framework [11, 35] provides trace-driven infrastructure for packet-level simulations of traffic flow on HPC networks. It is built upon ROSS [14], a parallel discrete event simulation (PDES) engine, and has been successfully deployed for studying multi-job workloads on HPC networks [27, 43]. The optimistic nature of the ROSS PDES engine drives the scalability of the TraceR-CODES framework and enables fast simulation using large core counts in comparison to other simulation frameworks.

CODES provides a unified API for simulating traffic flow on many HPC networks, e.g. dragonfly, torus, express mesh, and fat-tree. The network being simulated can be selected at runtime along with other parameters such as the topology dimensions, link bandwidth, routing scheme etc. The default fat-tree network in CODES assumes a single rail, single plane topology. For this work, we have augmented the default model with parameter-based instantiation of various configuration options discussed in Section 2. In the new model, a user can specify the number of rails and planes. Likewise, tapering can be done at leaf switches.

4.1 Advances in TraceR

TraceR is a parallel execution trace simulator that replays the control and communication flow of HPC applications on top of CODES. For multi-job workloads, each application is concurrently simulated and shares network resources with other applications. Users can also specify the job placement and task mapping for each application. For compute regions of the application, TraceR allows replacing and scaling the execution time recorded in the traces. This enables predictions of likely scenarios on future systems with different computational power.

Low-effort trace collection: A practically useful method for collecting traces should require minimal changes to the application, incur low overhead, and be memory efficient. While BigSim trace collection using AMPI [25] incurs low overhead, it requires recompilation of all the parallel libraries used by the application, and thus can result in significant effort.

Recent support for the Open Trace Format (OTF2) [6] by MPI tracing tools such as ScoreP [2] and TAU [37] provides an easy-to-use alternative for collecting application traces. Since these tools use the PMPI interface to intercept MPI calls, they can be added at link time. Thus, we added the capability to use OTF2 traces for simulating application's flow in TraceR. Because of this addition, we are able to work with application teams and obtain traces for large applications with low effort.

MPI collectives: Section 3 showed that collective operations are used in many applications for exchanging data among MPI processes and synchronizing the processes. In the past, TraceR has supported tree-based implementations of three collectives: MPI_Bcast, MPI_Reduce, and MPI_Barrier.

We have added the following collectives that are also used in many applications: MPI_Allreduce, MPI_Alltoall(v), and MPI_Allgather. For each of these operations, we have implemented several different algorithms that are used in MPI libraries based on the size of messages and that of the communicators [40]. For example, the Bruck algorithm, multi-pair send-recv, and ring algorithms have been implemented for MPI_Alltoall using small, medium, and large message sizes, respectively. To the best of our knowledge, such detailed modeling of collective algorithms is not supported in any other simulator.

Eager-Rendezvous protocol: Different protocols are used by MPI to perform communication among individual processes based on the size of messages. This not only impacts the amount of active communication on the network, but can also affect the way different MPI processes are synchronized [17]. To this end, we have added support for *eager* and *rendezvous* protocols in TraceR.

The addition of these features, along with the existing capabilities viz. parallel scalability, packet-level traffic flow, and support for multi-job workloads, makes the TraceR-CODES framework highly suitable for application simulations with low effort.

4.2 Validation

The TraceR-CODES simulation framework has been verified and validated in several past publications [27, 35]. In this study, for all configurations of fat-tree, we have conducted controlled simulations of micro-benchmarks, e.g. ping-pong, to verify that the simulations behave as expected. We have also compared predicted performance with observed performance for several micro-benchmarks and applications. Due to lack to space, we present results for a representative set in Figures 4 & 5. These results span a wide range of message sizes and scenarios of interest viz. latency-bound and bandwidth-bound executions.

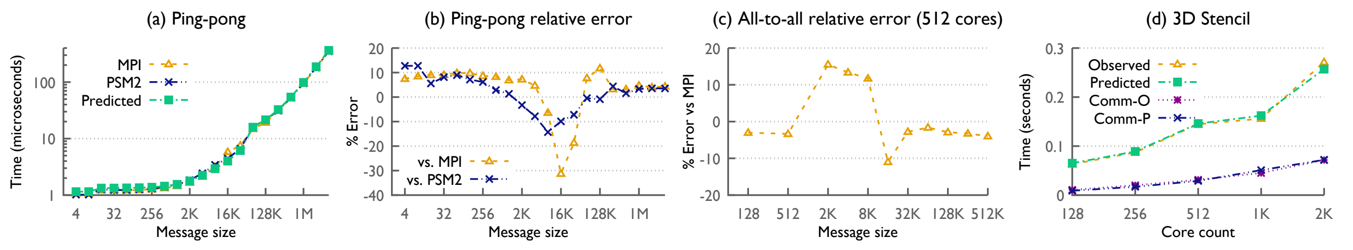

Figure 4: Comparing prediction results from TraceR-CODES framework with results from executions on a production system, Quartz [7].

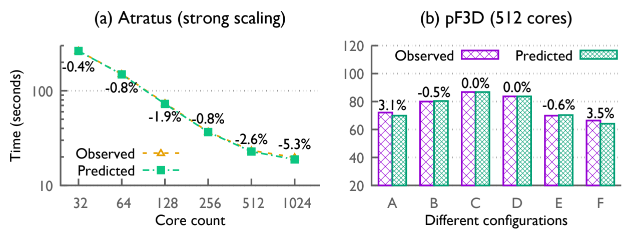

Figure 5: Prediction accuracy is high for latency-bound Atratus and bandwidth-bound pF3D.

In the validation experiments, observed performance was recorded from executions using MPI versions of different codes on Quartz, a 2:1 tapered fat-tree system with 100 Gb/s link bandwidth [7]. The radix of each switch is 48, with 32 nodes connected to each switch at the leaf level. Quartz was used for collecting traces of codes with computation (3D Stencil, Atratus, pF3D) to prevent mispredictions due to differences in compute time while another system (Catalyst [3]) was used for traces of communication-only micro benchmarks.

Figures 4(a)-(c) present the prediction accuracy for *ping-pong* and *all-to-all* micro-benchmarks. In the ping-pong benchmark, two MPI processes are run on two nodes (one per node) and messages are exchanged among them. In the all-to-all experiments, 512 MPI processes running on 16 nodes invoke `MPI_Alltoall` calls repeatedly. For a wide range of message sizes (4 bytes to 4 MB), we find that the predicted execution time is similar to the observed time. Higher error in comparison to MPI is observed at medium message sizes where the performance of the MPI implementation is erratic. However, for the same data points, predictions are much closer to results obtained for PSM2, the communication layer directly used by MPI on Quartz.

In 3D Stencil, MPI processes are arranged in a three-dimensional grid. In every iteration, each MPI process exchanges messages with six nearest-neighbors and performs a 7-point stencil update on the data it owns. For 3D Stencil, the predicted execution time closely follows the trend of the observed time as the core count is increased. Figure 4(d) shows accuracy results for two versions of 3D Stencil – with and without computation.

Figure 5(a) shows that for a latency-bound application such as Atratus (Section 3), high prediction accuracy is observed for a strong scaling run. Up to 256 cores, MPI time for different processes in Atratus is between 11% and 34% of the total run time, while it is between 31% and 63% on 512 and 1024 cores.

Results for bandwidth-bound pF3D (Section 3) are shown in Figure 5(b). It spends 20–25% of its execution time in communication. Predictions for pF3D are within 4% of the observed time for various configurations that differ in task mapping and number of processes per node. To the best of our knowledge, validation results for such applications have not been shown for any other simulator.

Along with past studies [27, 35], these results demonstrate that the TraceR-CODES framework predicts trends of the execution time of benchmarks and production applications accurately for the fat-tree topology.

5 COMPARATIVE ANALYSIS

In this section, we use TraceR to understand the performance trade-offs of different design options (Section 2) that represent future networks. In order to do so, we design prototype systems with ~4,500 nodes, which is similar to the expected node count of Summit and Sierra.

The six prototype systems used in this study are based on the configurations in Table 1, which are currently offered by network vendors: 1) SR-EDR: single rail single plane EDR, 2) DRP-T-EDR: dual rail dual plane tapered EDR, 3) DRP-EDR: dual rail dual plane EDR, 4) SR-HDR: single rail single plane HDR, 5) DR-T-HDR: dual rail single plane tapered HDR, 6) DR-HDR: dual rail single plane HDR. SR and DR refer to single rail and dual rail respectively. If 'P' appears in the configuration name, it suggests dual plane and if 'T' appears, it reflects tapering of the network.

The application traces used in these simulations are generated on 16,384 MPI processes using the same input problems that were used in the results shown in Section 3. PMPI-based ScoreP library is used for collecting OTF2 traces, which leads to less than 5% tracing overhead. In all simulations, a 3-level fat-tree is constructed using routers with 36 ports and 90 ns router delay. The packet size is fixed at 4 KB and FTREE static routing scheme is used. These parameter choices have been made because they resemble the settings used on current production systems.

5.1 Communication-only Simulations

The first set of results we present are for communication-only scenarios, i.e. application traces are simulated with all the computation regions ignored (not simulated). These results help understand the

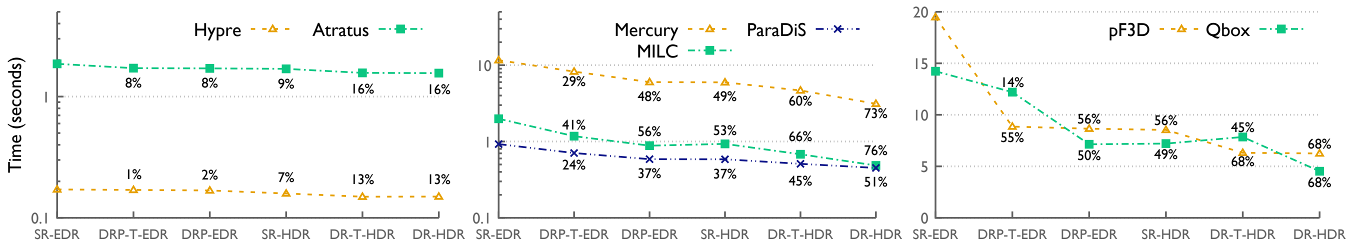

Figure 6: Comparing communication-only scenario for applications using 16,384 processes and 1,024 nodes. % performance gains in comparison to SR-EDR are shown. Table 1 lists the network configurations.

direct effect of network configurations on application communication patterns and provide an upper bound on the performance benefits of different configurations. In these runs, 16,384 processes are simulated on 1,024 nodes (~23% of system size) with 16 processes per node allocated using a linear placement policy. Figure 6 presents the results for all the applications along with performance improvements relative to single rail single plane EDR (SR-EDR).

Hypre and Atratus: For these applications that predominantly use small and medium sized messages for point-to-point and collective communication, the performance impact of changing the network configuration is low (< 17%). For Atratus, providing additional bandwidth at the lower levels of fat-tree either via use of dual rail (e.g. SR-EDR vs. DRP-T-EDR) or via use of links with higher bandwidth (e.g. DRP-T-EDR vs. DR-T-HDR) improves performance because of its medium size messages. However, since these messages are mostly communicated with processes that are close in MPI rank space, tapering does not affect performance when linear mapping is used (e.g. DRP-T-EDR vs. DRP-EDR).

For Hypre, only when HDR links are used, we observe a performance impact of network configurations. This is because the average and maximum message sizes of Hypre are lowest among all the applications and thus Hypre is bound by per message overheads on EDR networks.

Overall, we find the potential gains of using higher capability fat-tree configurations limited for relatively *light* communication patterns in Hypre and Atratus.

Mercury, MILC, and ParaDiS: All these applications are impacted in a similar way by varying network configurations: significant improvement is obtained by adding more bandwidth and tapering leads to a noticeable loss in performance. For Mercury, this is expected since it makes heavy use of collectives such as `MPI_Allreduce`, `MPI_Bcast`, and `MPI_Reduce` with large message sizes over large communicators. In addition, processes far away in MPI rank space also exchange many medium size messages.

For MILC, every MPI process communicates with very few MPI processes. However, these exchanges are typically over medium to large sized messages. Due to the 4D layout of MPI processes, some of the communicating pairs are far apart in the MPI rank space. These reasons account for the performance impact observed for MILC. An interesting observation for these results is marginally lower performance of single rail HDR (SR-HDR) in comparison to dual rail dual plane EDR (DRP-EDR), although they offer similar total bandwidth. We suspect this is because MILC sends very few large sized messages, and thus delays caused by routers and links can add up. Since, DRP-EDR has more routers and links in comparison to SR-HDR, fewer packets traverse per router/link and thus the impact of those delays is lower.

ParaDiS generates many small and medium sized message sends, but it also performs many large sized message exchanges, and calls `MPI_Allreduce`. As a result, the network configuration has a larger impact on the performance of ParaDiS than Hypre and Atratus.

pF3D and Qbox: These applications make heavy use of collectives and thus show up to 68% improvement with the best fat-tree configuration as shown in Figure 6(right). For pF3D, both the `MPI_Alltoall` calls and the point-to-point calls are among processes relatively close to one another in the MPI rank space. Thus, tapering the network does not impact performance significantly (e.g. DRP-T-EDR vs. DRP-EDR).

For Qbox, while the `MPI_Alltoall` communication is among "nearby" processes, other collectives with large message sizes occur over processes spread all over the MPI rank space. Thus, it not only benefits from higher bandwidth at the lower levels, but also shows better performance with full fat-tree configurations. An interesting data point for Qbox is the transition from SR-HDR to DR-T-HDR. Since in dual rail single plane HDR, fewer nodes are connected per leaf switch (12 nodes per switch) in comparison to SR-HDR (18 nodes per switch), more data has to traverse through higher levels in the tree when processes communicate with far away processes in MPI rank space. This negatively impacts the gains provided by DR-T-HDR, and results in lower performance.

In summary, with communication patterns that include large message sized exchanges among even few pairs of far away neighbors or include large sized collectives, significant gains are observed with higher network bandwidth (dual rail and/or HDR links) and full fat-tree configurations (DRP-EDR, DR-HDR).

5.2 Varying Computation Scaling

In the previous section, we studied the impact of network configurations on executions with the computation turned off. We now study cases in which computation is also considered alongside communication. These cases are important because the performance of an application can change significantly due to the interplay of computation and communication. Hence, such an analysis can help find the right balance between computational and communication resources.

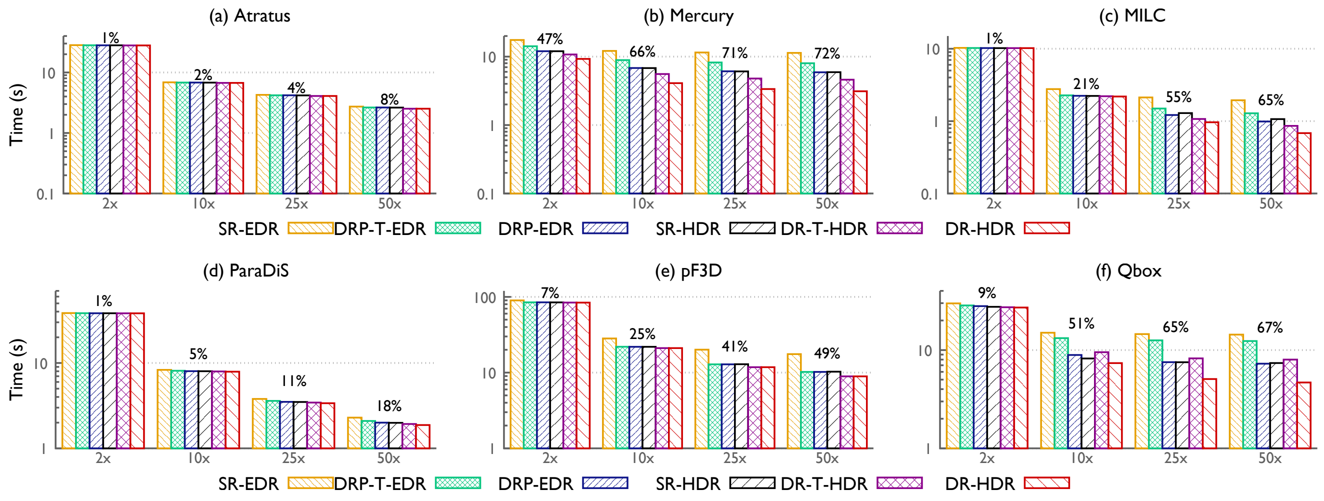

Figure 7: Analyzing the effect of network configurations on scenarios with different computation scaling. % difference between the best and the worst configuration is shown.

In our simulation framework, cores (or logically equivalent computational resources) are separate entities that interact with the NIC/network as is the case on real systems. Thus, if communication-computation overlap is possible, it is simulated appropriately. Note that TraceR does not simulate or model computation, it only fast-forwards the clock when a computation region is encountered.

In these experiments, the computational capability and execution time on an Intel Xeon CPU E5-2695v4 operating at 2.1 GHz with one socket is taken as the base value. This CPU has a peak flop/s rating of 600 GFlop/s. The execution time of compute regions is scaled by 2×, 10×, 25×, and 50× relative to this CPU to conduct the analysis for future systems. This implies that the 2× scaling results assume that on a future node, the computation can be performed twice as fast compared to this CPU.

We use the 2× case to represent systems similar to current clusters but with additional CPU sockets, while the 10× case is used as representative of future systems with accelerators. The remaining cases represent either more compute-intensive systems or applications that can exploit most of the computational power of future nodes.

Figure 7 shows the predicted impact of network configurations with different levels of computation scaling. Results on Hypre are omitted because the impact of network configurations on relative gains is similar to that on Atratus. With 50× scaling, the predicted results are similar to the results obtained for communication-only scenarios. Hypre and Atratus have minimal gains, pF3D gains from increased bandwidth at the lower level of the fat-tree but is not impacted by tapering, and Mercury, MILC, and Qbox improve with most changes in configurations. For ParaDiS though, even at 50× scaling, the potential impact of network configurations is much lower in comparison to the communication-only scenario.

The biggest difference for 50× scaling in comparison to the communication-only scenario is for ParaDiS where load imbalance among processes and similar time spent in computation and communication reduces the best predicted improvement from 51% to 21%. For both Atratus and pF3D, similar reduction in benefits from a better network configuration is observed due to the presence of relatively significant computation time. This trend continues for the case with 25× scaling.

As the computation scaling is decreased further to 10×, for many applications we observe a significant change in the impact of network configurations. For Hypre, Atratus, and ParaDiS, the computation time begins to dominate the total execution time and the impact of network configuration is limited. For MILC and pF3D, only the SR-EDR configuration is now significantly worse than other configurations, and the performance improvement due to a more capable network is at most 25%. For both these applications, the change in absolute time between various network configurations (from DRP-T-EDR to DR-HDR) is similar to the 50× scenario. However, with 10× scaling, since the time spent in computation is twenty times the time spent in communication, the net effect of these improvements is marginal.

For Mercury and Qbox, even at 10× scaling, the time spent in communication is dominant. The impact of adding more bandwidth and using full fat-tree systems instead of a tapered one is similar to earlier cases, and the gains are proportionate. Finally, at 2× scaling, Mercury is the only application whose time spent in computation is not significantly greater than the communication time.

5.3 Effect on Strong Scaling

In these last sets of results for single job simulations, we study the impact of network configurations on strong scaling. While system capabilities continue to grow, the size of input problems is not increasing proportionately in some domains. Hence, strong scaling is an important characteristic to consider for future systems.

In our experiments, strong scaling is achieved by simulating execution of jobs with 16,384 MPI processes (used earlier) on three different node counts: 256, 1,024, and 4,096. This results in allocation of 64, 16, and 4 MPI processes per node. As before, we follow a simple model of linearly scaling the computation assuming

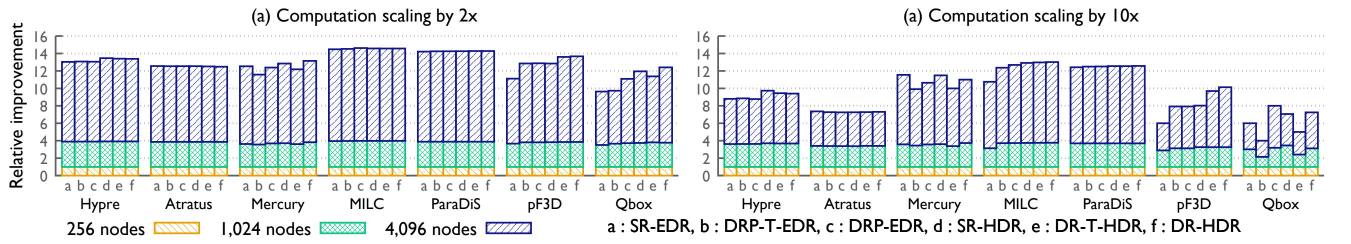

Figure 8: Impact of network configuration on strong scaling a problem by simulating 16,384 MPI processes on different node counts. For each configuration, improvement shown for an application is speed up relative to the application's performance on 256 nodes using that network configuration.

within-node parallelization. We study these cases for two different computational scaling scenarios from the previous section: 2× and 10×.

For any given node count, we find that the impact of network configurations on different applications is similar to the results described in the previous section. This is not unexpected because strong scaling a problem with linear scaling of computation does not change its computation-to-communication ratio. Thus, in this section, we focus on comparing predicted speedups when applications are scaled from 256 to 1,024 nodes as shown in Figure 8. The y-axis in this figure plots the predicted speedup for individual applications relative to their performance on 256 nodes for the same network configuration. The height of each bar shows the additional speedup over the data point right below it.

For the set with 2× scaling of compute regions relative to current systems (Figure 8(a)), we observe that the impact of network configurations on speedup for many applications is minimal. Only for pF3D and Qbox, dual rail/plane and HDR configurations lead to better scaling in comparison to the scaling on the single rail single plane EDR system (up to 27% gain in speedup). These results also show that communication bottlenecks do not prevent efficient scaling of most applications.

For the case with 10× computation scaling as shown in Figure 8(b), the impact of network configurations on scaling is significantly higher. First, the speedups are relatively lower for all network configurations, especially SR-EDR. Second, many applications (Hypre, MILC, pF3D, and Qbox) show better scaling with more capable network configurations. Third, for Qbox and Mercury, scaling is better for full fat-tree configurations in comparison to tapered configurations.

In summary, our predictions suggest that with 2× computation scaling, network performance has limited impact on scaling of applications, but with 10× computation scaling, careful consideration of network configurations is needed to obtain good strong scaling behavior.

6 ANALYZING MULTI-JOB WORKLOADS

Simultaneous execution of multiple large jobs that share network resources is a common production scenario. In this section, we study the impact of such workloads on performance of individual jobs on different fat-tree configurations. We also compare the overall performance of workloads across different configurations.

The multi-job workloads used in these experiments consist of 17 jobs of 4,096 processes each running on 256 nodes. Each job is arbitrarily assigned one of the applications described earlier. Since the simulations of Hypre and Atratus have shown similar behavior so far, we do not use Hypre in these experiments in order to focus on a diverse set. Trace generation was done on Quartz by running weak-scaled versions of the input problems (Table 2) on 4K MPI processes. We simulate three such randomly generated workloads, in which each application is assigned to ~8 jobs in total.

In the first set of simulations of multi-job workloads, we use 2× computation scaling. Nodes are allocated to jobs using a "randomized clusters" policy: jobs are assigned nodes in small clusters spread throughout the system. For these experiments, we find that the average performance of most applications (except Mercury) across different network configurations is similar (<5% difference). This is expected since the time spent in computation is >90% for all these applications. For Mercury, we observe up to 33% gains; these results are similar to the results observed for Mercury in the previous section. We also find that the impact of inter-job interference is minimal given the computation-heavy nature of the workloads.

Figure 9(a) presents the average time spent in executing a time step of different applications when running in a multi-job workload. For these results, 10× computation scaling is used, and the job placement policy is the same as before. Both Atratus and ParaDiS are not significantly impacted by changes in network configurations, even when part of a multi-job workload.

Mercury and Qbox show performance gains with every improvement in network configuration. The trends and impact of individual configuration changes are similar to the single-job scenarios studied earlier in Figure 7. Note that for Qbox, DRP-T-HDR configuration performs better than SR-HDR in this case because the randomized clusters job placement eliminates the benefit of having more nodes per leaf switch. With nodes spread throughout the system, more traffic has to traverse to higher level links for the SR-HDR configuration in comparison to the single job scenario with linear placement.

For MILC and pF3D, we observe higher gains (31% and 51%) when better network configurations are used in comparison to the gains predicted for single-job scenarios (21% and 25%). The other difference from single-job results in Figure 7 is that in addition to the use of dual rail EDR, using a full fat-tree configuration and increasing link bandwidth (EDR to HDR) also helps. This is because

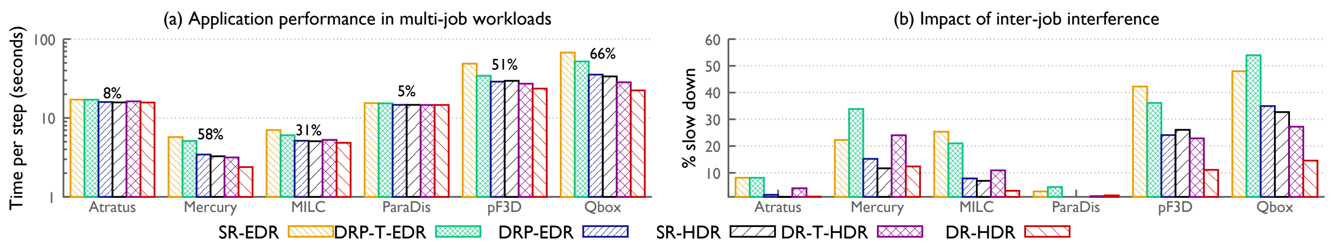

Figure 9: Analyzing the performance of applications in a multi-job workloads. Each workload consists of 17 jobs, each allocated 256 nodes and is simulated with 10× computation scaling. For (b), the percentage slow down shown in relative to predictions without interference.

communicating pairs that are nearby in the linear placement are spread out in the randomized clusters placement and hence these applications also make use of links at higher levels of the fat-tree.

Figure 9(b) presents the average percentage slow down that is observed for individual applications relative to their performance for 256 nodes interference-free single-job runs. For the two configurations with lower total bandwidth (SR-EDR and DRP-T-EDR), the effect of inter-job interference is higher. We also find that pF3D and Qbox are impacted more by inter-job interference than other applications. This is most likely due to their heavy use of tightly-coupled `MPI_Alltoall(v)` operations. For some applications, the tapered configurations show higher impact of inter-job interference.

In Figure 10, we compare the impact of network configuration on total performance of the multi-job workloads. The values shown here are an average over the three multi-job workloads we have run. The total performance of a workload with n jobs is computed as $P_w = \sum_{i=0}^{n-1} P_i^{norm}$, where P_i^{norm} is the normalized execution rate (number of steps completed per unit time) for an application across different network configurations. The rates are normalized based on the rate observed for the best performing network configuration for a given job. This implies that P_i^{norm} for job i is always less than or equal to one, and the total performance of a workload is bounded by the number of jobs in the workload.

Figure 10 shows that for the multi-job workloads simulated using the randomized clusters job allocation policy, higher gains are observed when transitioning from dual rail dual plane tapered EDR (DRP-T-EDR) to dual rail dual plane EDR (DRP-EDR). We notice a similar transition in dual rail HDR-based configurations. We find improvements to be marginal when transitioning from single rail HDR to dual rail tapered HDR for the input problems simulated in these multi-job workloads.

Cost-performance tradeoffs: From a procurement point of view, the performance improvements discussed above should be combined with the expected dollar costs to analyze the performance-cost tradeoffs. Consider a scenario in which the normalized costs (w.r.t. SR-EDR) of the network configurations and the total cost of the system are as shown in Table 5. Here, we assume network cost is 10% of the total system cost for SR-EDR. These estimates are based on past procurements and the best data currently available to us.

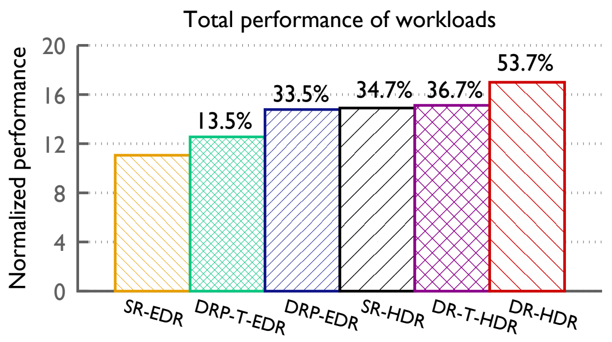

Figure 10: Comparing the total normalized performance of workloads executed on different network configurations with 10× computation scaling.

Table 5 also combines the cost estimates with the results in Figure 10 and computes the predicted cost to performance value of different fat-tree configurations. We find that the cost-to-performance ratio drops rapidly from SR-EDR to DRP-T-EDR to DRP-EDR, i.e. dual rail EDR configurations provide better returns for their cost in comparison to the single rail EDR configuration. Further, the cost-to-performance ratio of all HDR configurations is lower (i.e. better) than the EDR configurations. In particular, the SR-HDR configuration provides similar performance as DRP-EDR at a much better cost to performance ratio.

7 RELATED WORK

Several topologies have been proposed and used in HPC networks, e.g. hypercube [24], fat-tree [30], torus [19, 33], dragonfly [21, 29]. Among these topologies, variations of the fat-tree and dragonfly topologies are currently being used in many systems. This paper focuses on fat-tree because of the multiplicity of design options available for it [20, 34].

For analysis and prediction of performance on HPC networks, several approaches ranging from analytical modeling [12, 26, 36, 38] to flit and packet-level simulations [18, 28, 35, 41] have been proposed. We deploy trace-driven packet-level simulations because they allow for use of existing codes for simulating their communication complexity.

Table 5: Cost and performance summary for different fat-tree configurations. Total cost is computed assume network cost is 10% of the total cost for SR-EDR. Lower cost/performance is better.

Configuration	Network cost	Total cost	Total performance	Total cost/ Performance
SR-EDR	1	1	1	1
DRP-T-EDR	1.1	1.01	1.14	.89
DRP-EDR	1.79	1.08	1.34	.81
SR-HDR	1.4	1.04	1.35	.77
DR-T-HDR	1.67	1.07	1.37	.78
DR-HDR	2.81	1.18	1.54	.77

Existing work most closely related to our work includes [42], [32], [34], and [27]. Wolfe et al. use simulations to study the impact of multiple rails and planes on communication-only traces of miniapplications [42]. In [32], Leon et al. study the impact of tapering a production fat-tree network on applications by turning off half the core switches in the network. Michelogiannakis et al. [34] study the impact of task mapping on performance of miniapplications on a tapered fat-tree system. In [27], simulations are used to compare different topologies for several miniapplications based workloads.

The work presented in this paper differs from existing work in several ways. First, we use traces generated from production applications and libraries. Second, computation and its interplay with communication is considered using simple modeling of compute time. Third, a number of fat-tree configurations (with different link bandwidths, rails, planes, and tapering) are compared. Finally, we present MPI-specific extensions (support for collective operations and messaging protocols) to a current simulation framework that reduces the gap between simulation and real world executions.

8 CONCLUSION

The decision making process for system procurement in HPC centers is of critical importance due to the magnitude of investment required for acquiring and operating supercomputers. Increasing dollar cost of networks presents a challenge in this regard as it can impact the available budget for buying other components of the system such as nodes and memory. As a result, HPC centers can benefit from methodologies that can help understand the impact of reduced network capabilities to mitigate the risks posed by it.

In this paper, we have demonstrated that predictions derived from a reliable simulation framework can assist in estimating the performance impact of design options available in networks. In particular, we have shown that the TraceR-CODES framework is capable of simulating complex applications and their workloads. Accuracy of predictions made using these simulations has been evaluated using micro-benchmarks and applications.

Using the TraceR-CODES framework, we presented results on the potential impact of different fat-tree configurations on production applications and libraries. In Section 5.1, we learned that increasing total bandwidth using dual rails/planes and high bandwidth links significantly reduces the time spent in communication for many applications if computation is ignored. But, if the communication pattern of an application consists only of small sized messages, higher bandwidth is unused.

When computation is also considered in simulations (Section 5.2), the impact of network configurations on overall application performance varies. For future systems with computation capabilities similar to the systems of today, limited gains are predicted for most applications (even those with high communication volume). For applications simulated in this paper, results on systems with higher computational capabilities indicate that significant gains can be obtained by use of dual rail/plane EDR networks for communication-intensive applications. The benefits of dual rail HDR networks were found to be significant only when computation can be scaled by 50× for the input problems simulated in this paper.

For multi-job workloads simulated in this paper (Section 6), we found that dual rail/plane EDR network can significantly reduce the impact of inter-job interference in comparison to single rail EDR network. The additional impact of dual rail tapered HDR system is predicted to be much lower. Figure 10, which summarizes our findings for expected total throughput of workloads, shows a similar trend. Overall, we find that the HDR-based networks are expected to provide high performance with a relatively low and better cost-to-performance ratio (Table 5).

In the future, we plan to extend this work in two directions. First, we plan to study the role of sophisticated job placement and task mapping schemes for different network configurations. While currently not commonly deployed in production, especially on fat-tree networks, we anticipate that topology-aware job placement and task mapping can help mitigate the impact of reduced network capabilities. Second, we will study the impact of changing the number of MPI end-points per node on the performance observed on different network configurations.

REFERENCES

[1] 2015. Performance Scaled Messaging 2 (Intel). https://www.intel.com/content/dam/support/us/en/documents/network/omni-adptr/sb/Intel_PSM2_PG_H76473_v1_0.pdf. (2015).

[2] 2015. Score-P User Manual. https://silc.zih.tu-dresden.de/scorep-current/pdf/scorep.pdf. (2015).

[3] 2017. Catalyst (LLNL). https://hpc.llnl.gov/hardware/platforms/catalyst. (2017).

[4] 2017. MFEM: Modular finite element methods. mfem.org. (2017).

[5] 2017. MULARD. (2017). https://codesign.llnl.gov/mulard.php.

[6] 2017. Open Trace Format 2. (2017). https://silc.zih.tu-dresden.de/otf2-current/index.html.

[7] 2017. Quartz (LLNL). https://hpc.llnl.gov/hardware/platforms/Quartz. (2017).

[8] 2017. The Sierra Advanced Technology System. http://computation.llnl.gov/computers/sierra-advanced-technology-system. (2017).

[9] 2017. Stampede (TACC). https://www.tacc.utexas.edu/stampede/. (2017).

[10] 2017. Summit (OLCF). http://www.olcf.ornl.gov/summit. (2017).

[11] Bilge Acun, Nikhil Jain, Abhinav Bhatele, Misbah Mubarak, Christopher D. Carothers, and Laxmikant V. Kale. 2015. Preliminary Evaluation of a Parallel Trace Replay Tool for HPC Network Simulations. In *Proceedings of the 3rd Workshop on Parallel and Distributed Agent-Based Simulations (PADABS '15).* LLNL-CONF-667225.

[12] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. 1995. LogGP: incorporating long messages into the LogP modelâĂŤone step closer towards a realistic model for parallel computation. In *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures (SPAA '95).* ACM, New York, NY, USA, 95–105. https://doi.org/10.1145/215399.215427

[13] A. Arsenlis, W. Cai, M. Tang, M. Rhee, T. Oppelstrup, G. Hommes, T. G. Pierce, and V. V. Bulatov. 2007. Enabling strain hardening simulations with dislocation dynamics. *Modelling and Simulation in Materials Science and Engineering* 15, 6 (2007).

[14] David W. Bauer Jr., Christopher D. Carothers, and Akintayo Holder. 2009. Scalable Time Warp on Blue Gene Supercomputers. In *Proceedings of the 2009*

ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation (PADS '09). IEEE Computer Society, Washington, DC, USA.

[15] Claude Bernard, Tom Burch, Thomas A. DeGrand, Carleton DeTar, Steven Gottlieb, Urs M. Heller, James E. Hetrick, Kostas Orginos, Bob Sugar, and Doug Toussaint. 2000. Scaling tests of the improved Kogut-Susskind quark action. *Physical Review D* 61 (2000).

[16] Patrick S. Brantley, Shawn A. Dawson, Michael Scott McKinley, Matthew J. O'Brien, David E. Stevens, Bret R. Beck, Eric D. Jurgenson, Chris A. Ebbers, and James M. Hall. 2013. Recent Advances in the Mercury Monte Carlo Particle Transport Code. In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C'13).* Sun Valley, ID.

[17] Ron Brightwell and Keith Underwood. 2003. Evaluation of an eager protocol optimization for MPI. In *European Parallel Virtual Machine/Message Passing Interface UsersâĂŹ Group Meeting.* Springer, 327–334.

[18] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. 2014. Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms. *J. Parallel and Distrib. Comput.* 74, 10 (June 2014), 2899–2917.

[19] Dong Chen, N.A. Eisley, P. Heidelberger, R.M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D.L. Satterfield, B. Steinmacher-Burow, and J.J. Parker. 2011. The IBM Blue Gene/Q interconnection network and message unit. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for.* 1–10.

[20] Salvador Coll, Eitan Frachtenberg, Fabrizio Petrini, Adolfy Hoisie, and Leonid Gurvits. 2003. Using multirail networks in high-performance clusters. *Concurrency and Computation: Practice and Experience* 15, 7-8 (2003), 625–651.

[21] Greg Faanes, Abdulla Bataineh, Duncan Roweth, Tom Court, Edwin Froese, Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, and James Reinhard. 2012. Cray Cascade: A Scalable HPC System Based on a Dragonfly Network. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12).* IEEE Computer Society Press, Los Alamitos, CA, USA.

[22] R.D. Falgout, J.E. Jones, and U.M. Yang. 2006. The Design and Implementation of hypre, a Library of Parallel High Performance Preconditioners. In *Numerical Solution of Partial Differential Equations on Parallel Computers*, A.M. Bruaset and A. Tveito (Eds.). Vol. 51. Springer-Verlag, 267–294.

[23] F. Gygi, E. W. Draeger, B. R. De Supinski, R. K. Yates, F. Franchetti, S. Kral, J. Lorenz, C. W. Ueberhuber, J. A. Gunnels, and J. C. Sexton. 2005. Large-Scale First-Principles Molecular Dynamics Simulations on the Blue Gene/L Platform using the Qbox Code. *In Proceedings of Supercomputing 2005* 4 (2005), 24. Conference on High Performance Networking and Computing, Gordon Bell Prize finalist.

[24] John P. Hayes, Trevor N. Mudge, and Quentin F. Stout. 1986. Architecture of a Hypercube Supercomputer. In *ICPP.* 653–660.

[25] Chao Huang, Gengbin Zheng, Sameer Kumar, and Laxmikant V. Kalé. 2006. Performance Evaluation of Adaptive MPI. In *Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming 2006.*

[26] Nikhil Jain, Abhinav Bhatele, Xiang Ni, Nicholas J. Wright, and Laxmikant V. Kale. 2014. Maximizing Throughput on a Dragonfly Network. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14).* IEEE Computer Society. LLNL-CONF-653557.

[27] Nikhil Jain, Abhinav Bhatele, Samuel T. White, Todd Gamblin, and Laxmikant V. Kale. 2016. Evaluating HPC Networks via Simulation of Parallel Workloads. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '16).* IEEE Computer Society. LLNL-CONF-690662.

[28] Nan Jiang, Daniel U. Becker, George Michelogiannakis, James Balfour, Brian Towles, John Kim, and William J. Dally. 2013. A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator. In *IEEE International Symposium on Performance Analysis of Systems and Software.*

[29] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. 2008. Technology-Driven, Highly-Scalable Dragonfly Topology. *SIGARCH Comput. Archit. News* 36 (June 2008), 77–88. Issue 3.

[30] C.E. Leiserson. 1985. Fat-trees: Universal Networks for Hardware-Efficient Supercomputing. *IEEE Transactions on Computers* 34, 10 (October 1985).

[31] Charles E. Leiserson. 1985. Fat-trees: Universal Networks for Hardware-efficient Supercomputing. *IEEE Trans. Comput.* 34, 10 (Oct. 1985), 892–901.

[32] Edgar A. Leon, Ian Karlin, Abhinav Bhatele, Steven H. Langer, Chris Chambreau, Louis H. Howell, Trent D'Hooge, and Matthew L. Leininger. 2016. Characterizing Parallel Scientific Applications on Commodity Clusters: An Empirical Study of a Tapered Fat-tree. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '16).* IEEE Computer Society, Article 78, 12 pages. http://dl.acm.org/citation.cfm?id=3014904.3015009 LLNL-CONF-681011.

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

[34] George Michelogiannakis, Khalid Ibrahim, Jeremiah Shalf, John anWilke, Samuel Knight, and Joseph Kenny. 2017. APHiD: Hierarchical Task Placement to Enable a Tapered Fat Tree Topology for Lower Power and Cost in HPC Networks. *CCGrid 2017 (to appear)* (2017).

[35] Misbah Mubarak, Christopher D. Carothers, Robert B. Ross, and Philip Carns. 2016. Enabling Parallel Simulation of Large-Scale HPC Network Systems. *IEEE Trans. Parallel Distrib. Syst.* (2016). https://doi.org/10.1109/TPDS.2016.2543725

[36] Mohamed Ould-Khaoua and Hamid Sarbazi-Azad. 2001. An Analytical Model of Adaptive Wormhole Routing in Hypercubes in the Presence of Hot Spot Traffic. *IEEE Transactions on Parallel and Distributed Systems* 12, 3 (2001), 283–292.

[37] S. Shende and A. D. Malony. 2005. The TAU Parallel Performance System. *International Journal of High Performance Computing Applications, ACTS Collection Special Issue* (2005).

[38] Kyle L. Spafford and Jeffrey S. Vetter. 2012. Aspen: A Domain Specific Language for Performance Modeling. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12).* IEEE Computer Society Press, Los Alamitos, CA, USA, Article 84, 11 pages. http://dl.acm.org/citation.cfm?id=2388996.2389110

[39] C. H. Still, R. L. Berger, A. B. Langdon, D. E. Hinkel, L. J. Suter, and E. A. Williams. 2000. Filamentation and forward Brillouin scatter of entire smoothed and aberrated laser beams. *Physics of Plasmas* 7, 5 (2000), 2023–2032.

[40] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of Collective Communication Operations in MPICH. *International Journal of High Performance Computing Applications* 19, 1 (2005), 49–66.

[41] K.D. Underwood, M. Levenhagen, and A. Rodrigues. 2007. Simulating Red Storm: Challenges and Successes in Building a System Simulation. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS '07).*

[42] Noah Wolfe, Misbah Mubarak, Nikhil Jain, Jens Domke, Abhinav Bhatele, Christopher Carothers, and Rob Ross. 2017. Methods for Effective Utilization of Multi-Rail Fat-Tree Interconnects *(CCGrid 2017 (to appear)).*

[43] Xu Yang, John Jenkins, Misbah Mubarak, Robert B. Ross, and Zhiling Lan. 2016. Watch Out for the Bully! Job Interference Study on Dragonfly Network. In *Supercomputing.*

A ARTIFACT DESCRIPTION: [PREDICTING THE PERFORMANCE IMPACT OF DIFFERENT FAT-TREE CONFIGURATIONS]

A.1 Abstract

In this work, traces collected for several applications have been simulated using open source tools TraceR and CODES. We are currently working towards releasing the traces, network configuration files, and mapping files used in this work. For early access, please contact the lead author.

A.2 Description

A.2.1 Check-list.

- **Program: TraceR**
- **Compilation: using default options; compilation options do not affect the results.**
- **Data set: high level details are in the paper; we are currently working towards releasing the traces.**
- **Run-time environment: does not affect the results; needs MPI.**
- **Hardware: Catalyst (does not affect the results); needs MPI.**
- **Output: standard output from TraceR.**
- **Experiment workflow: collect traces, generate mappings, simulate.**
- **Experiment customization: none**
- **Publicly available: yes.**

A.2.2 How software can be obtained. All the features developed as part of this work are available in the current versions of TraceR and CODES, and can be obtained from the following locations:

- https://github.com/LLNL/tracer
- https://xgitlab.cels.anl.gov/codes/codes
- https://github.com/carothersc/ROSS

A.2.3 Hardware dependencies. None.

A.2.4 Software dependencies. TraceR, CODES, ROSS, MPI.

A.3 Installation

Standard installation process described in the documentation of ROSS, CODES, and TraceR have been followed.

A.4 Experiment workflow

The experiments in this paper follow a three step workflow:

- Design network prototype and identify required traces for simulations.
- Collect traces for described application.
- Generate mapping files and simulate using TraceR.

A.5 Evaluation and expected result

All the results used in this paper are based on the timing output from executions of TraceR. The users should be able to use the released traces, mapping files, and network configurations to reproduce them.

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-736289) and by Argonne National Laboratory under contract number DE-AC02-06CH11357.