Analyzing Cost-Performance Tradeoffs of HPC Network Designs under Different Constraints using Simulations

Abhinav Bhatele
bhatele@llnl.gov
Center for Applied Scientific Computing,
Lawrence Livermore National Laboratory
Livermore, California, USA

Nikhil Jain
nikhijain@nvidia.com
NVIDIA, Inc.
Santa Clara, California, USA

Misbah Mubarak
mmubarak@anl.gov
Mathematics and Computer Science Division,
Argonne National Laboratory
Lemont, Illinois, USA

Todd Gamblin
tgamblin@llnl.gov
Center for Applied Scientific Computing,
Lawrence Livermore National Laboratory
Livermore, California, USA

ABSTRACT

Identifying a suitable network topology and deciding its optimal configuration parameters are critical aspects of the overall HPC system design, procurement and installation process. Typically, multiple network topology choices are compared under the balanced injection-to-global bandwidth criterion to identify the best candidate. However, deviating from this balanced criterion may not impact application performance adversely and is often done in practice due to other considerations such as monetary cost. In this paper, we identify different practical constraints that determine the number of nodes, routers, and links, and in turn, influence dollar costs and impact network design. We design network topologies under one or more such constraints which represent different design points (iso-{*} analysis). We then perform a comprehensive, comparative evaluation of three scalable network topologies – dragonfly, express mesh, and fat-tree – enabled by parallel discrete-event simulations (PDES) of relevant HPC workloads. We identify network topologies that perform best under different iso-{*} configurations and compare their performance per dollar based on market data.

CCS CONCEPTS

• **Networks** → *Network simulations*; • **Computing methodologies** → *Discrete-event simulation.*

KEYWORDS

network design; discrete-event simulation; performance prediction; monetary cost

ACM Reference Format:
Abhinav Bhatele, Nikhil Jain, Misbah Mubarak, and Todd Gamblin. 2019. Analyzing Cost-Performance Tradeoffs of HPC Network Designs under Different Constraints using Simulations. In *SIGSIM Principles of Advanced Discrete Simulation (SIGSIM-PADS '19), June 3–5, 2019, Chicago, IL, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3316480.3325516

SIGSIM-PADS '19, June 3–5, 2019, Chicago, IL, USA

ACM ISBN 978-1-4503-6723-3/19/06…$15.00
https://doi.org/10.1145/3316480.3325516

1 MOTIVATION

High-speed interconnection networks are an integral part of high performance computing (HPC) systems, and determine the performance of many communication-heavy HPC applications. Hence, designing and/or identifying a suitable network topology and its optimal configuration parameters are critical aspects of HPC system design, procurement and deployment. Network designers, customers, and system administrators work under different constraints to decide the most suitable network topology and configuration for a specific deployment. Various factors such as monetary costs, system size (number of nodes), latency and/or bandwidth requirements come into play for different stakeholders. Customers may have a specific budget that they can spend on the entire system, a significant fraction of which is used to determine the number of nodes in the system. This might dictate the budget available for the network, which can become a constraint for network designers. On the other hand, end users are most concerned with the performance of their applications on the system. Together, these factors make a fair evaluation of different network topologies challenging.

Typically, networks are evaluated and compared under the *balanced* injection-to-global bandwidth constraint [9, 20]. A balanced configuration ensures a balance between injection bandwidth and global network bandwidth. Enforcement of the balanced criterion can put certain network topologies at a disadvantage when other practical constraints are also applied. However, in practice, deviating from a balanced configuration may not impact application performance adversely. For example, customers might not care for a balanced system because of budget constraints or their workloads might not require it. Network designers may not be able to build a balanced system due to a constraint on the number of nodes or to throttle injection bandwidth to mitigate congestion. Thus, in order to do a fair and practical comparison of different HPC networks, and to identify the strengths of different topologies under certain constraints, we approach the network design and configuration problem differently as described below.

Table 1: Assignment of router ports (radix = k) to nodes and links in order to ensure balance between injection and global network bandwidth. This also determines the maximum possible system size in terms of the number of nodes. Note: L refers to local links, and G refers to global links in a dragonfly topology.

Network topology	#nodes/router	#links/router	Maximum system size (#nodes)
All-to-all (A2A) dragonfly	k/4	k/2 (**L**), k/4 (**G**)	$(k/2+1)^2 \times (k/4+1) \times k/4$
Row-column (RC) dragonfly	k/6	2k/3 (**L**), k/6 (**G**)	$(k/3+1)^4 \times (k/6+1) \times k/6$
Express mesh (3D, gap=1)	k/4	3k/4	$(k/4+1)^3 \times k/4$
Fat-tree (three-level)	k/2	k/2	$k/2 \times k/2 \times k$

We identify different constraints that are typically applied when designing networks in practice, and describe these constraints in terms of the choices a network designer would make for the number of nodes, routers and links in the system. Each constraint fixes one or more quantities that define the system – number of nodes, routers, and/or links. We then build systems that use one of three scalable HPC network topologies – dragonfly [24] (two variations), express mesh [19] (a dense mesh-based network), and fat-tree [25], and apply the same constraints to them (we refer to this methodology as iso-{*} analysis in the rest of the paper). Note that we study two variations of the dragonfly topology – one has all-to-all links connecting *all* the routers in a group, implemented in the IBM PERCS system [6] and the Cray Slingshot network [2], and the other has all-to-all links connecting routers in each row/column in a group, implemented in the Cray Cascade (XC) system [15].

Each network and the corresponding system is constructed using five different *iso-X* configurations which results in twenty systems in total. Predicted performance on these systems is then compared using parallel discrete-event simulations (PDES) of the network models and replay of relevant HPC workloads composed of execution traces from different parallel application motifs. We find that different network topologies emerge as the best performers under different *iso-X* constraints. We also compare the performance per dollar of different systems using relative costs for the routers and links, derived from market data.

Most previous work [9, 19, 20, 28, 32] assume a balanced injection-to-global bandwidth constraint and then evaluate systems for that particular scenario. To the best of our knowledge, this is the first study that undertakes a comprehensive evaluation of network topologies across many practically relevant iso-{*} scenarios (Table 2). The novel contributions of this work are:

- Identifying different constraints that are applied when designing networks in practice, and describing these constraints in terms of the number of nodes, routers and/or links in the system (iso-{*} configurations).
- Design and analysis of HPC systems based on four different network topologies in each of five *iso-X* configurations using discrete-event simulations of relevant HPC workloads.
- Study of cost-performance tradeoffs of the designed systems and configurations using relative costs for routers and links, derived from market data.

2 HIGH-PERFORMANCE NETWORKS

In this section, for completeness, we describe three scalable, high-speed interconnection network topologies that have been proposed for use in HPC systems. For each network topology, we also describe the construction of a *balanced* configuration of the system. A balanced configuration ensures a balance between injection bandwidth and global network bandwidth.

Dragonfly: The dragonfly topology was proposed by Kim et al. [24], motivated by the arrival of high-radix routers in the market. This topology uses a set of high-radix routers to create a logical group which then connects with other groups giving the impression of a densely connected network. There are two ways in which routers within a group can be connected and we refer to them by different names in this paper. In an "all-to-all" (A2A) dragonfly, within a group, each router is connected to every other router by a direct link. In a "row-column" (RC) dragonfly, within a group, routers are arranged in logical rows and columns, and routers within each row and column are connected in an all-to-all fashion. The A2A dragonfly has been implemented by IBM in the PERCS system [6], and by Cray in the Shasta system (Slingshot network [2]). The RC dragonfly has been implemented by Cray in the Cascade (XC) systems [15]. The dragonfly topology is known to be highly scalable, i.e. very large systems can be constructed given a fixed router radix, without increasing the network diameter. However, especially for large systems, the need for adaptive non-minimal routing to tackle congestion and lack of shortest-path diversity can impact the observed performance on this network.

In a dragonfly network, some ports on each router are assigned to compute nodes, local links (connections to other routers in the same logical group), and global links (connections to routers in other groups). Assuming the radix of each router to be k, we can derive the optimal division of ports between compute nodes and network links (local and global) such that the network load is balanced. In the A2A dragonfly, in the shortest path between a source-destination pair, the maximum number of local links traversed is two for each global link and injection port (to which a compute node is connected). Hence, $k/4$ ports each should be assigned to compute nodes and global links, and $k/2$ ports should be assigned to local links to achieve a balanced configuration. This determines the maximum system size which is derived in the third column in Table 1. In the RC dragonfly, for each global link and injection port, a maximum of four local links are traversed when using shortest-path routing. Hence, $k/6$ ports each should be assigned to compute nodes and global links, and $2k/3$ ports should be assigned to local links to achieve a balanced configuration.

Express mesh: Express mesh is derived from an n-dimensional mesh topology by adding links within each dimension to reduce

the network diameter [19]. It is inspired by and a generalization of express cubes proposed by Dally [13]. In an n-dimensional mesh, each router is connected to $2 \times n$ other routers, two in each dimension. In an express mesh, within each dimension, additional links are used to connect a router to some or all routers (whose coordinates differ only in the specific dimension). The number of additional connections depends on a *gap* parameter, g. When $g = 1$, there are all-to-all connections between all routers that only differ in one mesh coordinate; in this case express mesh is the same as the HyperX topology [5].

A balanced construction of a three-dimensional (3D) express mesh with $g = 1$ necessitates assigning $k/4$ ports to compute nodes and the remaining $3k/4$ ports to network links (divided equally among all dimensions in a cuboidal shape). This is because the maximum number of hops in a 3D express mesh with $g = 1$ is three. Similar to n-dimensional mesh, express mesh is suitable for near-neighbor communication, but also provides high global bandwidth using the additional links. However, like dragonfly, lack of shortest-path diversity can negatively impact performance in express mesh networks also.

Fat-tree: The fat-tree topology was proposed by Leiserson in 1985 to connect parallel machines [25], and has been widely used in HPC systems of different sizes in the last three decades. The logical network represents an n-ary tree where the bandwidth between nodes of the tree increases as we get closer to the root. Since, commodity network routers have a fixed number of ports, the idea is implemented by grouping many routers together at the higher levels to give the impression of a large router with "fat" or high bandwidth links. These top-level routers in the fat-tree are typically referred to as *core* or *director-class* switches. This implementation of the fat-tree topology is also referred to as a Clos network.

In a balanced construction of this network, each leaf-level router assigns $k/2$ ports to compute nodes and the remaining $k/2$ ports are used to connect to routers at the next higher level. While fat-tree provides high bisection bandwidth, it requires additional routers as compared to other topologies, and thus, is typically more expensive. Further, it is not as scalable as the dragonfly network, i.e. for a given router radix, the largest system that can be constructed with a fat-tree is noticeably smaller than the dragonfly topology.

The balanced construction of each network determines the maximum possible system size in terms of the number of compute nodes or end-points that can be constructed using routers with a given radix (equations provided in the fourth column in Table 1). Figure 1 presents the largest system that can be constructed using routers with a certain number of ports and different network topologies. It is clear that dragonfly networks can support the highest number of nodes. At lower router radix, fat-tree and 3D express mesh can be used to build similar-sized systems but beyond a certain router radix, express mesh has an advantage.

3 DEFINING THE DESIGN SPACE

As described in Section 1, networks are typically evaluated and compared under the *balanced* injection-to-global bandwidth constraint [9, 19, 20, 28, 32]. In this work, we identify additional constraints that are often applied when designing and configuring

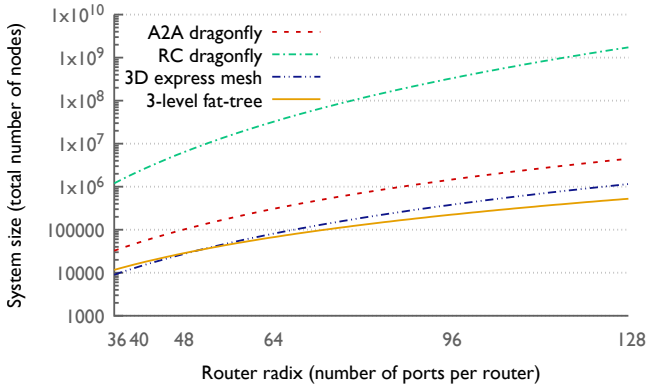

Figure 1: Plot showing the largest systems that can be built using different network topologies with increase in the number of ports per router.

networks in practice. We describe these constraint scenarios in terms of the choices network designers make with respect to the number of nodes, routers and links in the system.

When selecting and configuring an interconnection network for an HPC system, there is a large space of configurable parameters that includes the number of routers, number of links, number of nodes connected to a router, ratio of injection to link bandwidth, number of ports on each router etc. In this study, we choose number of nodes, routers and links as the three primary configurable parameters since they determine the system size. We fix the number of ports per router or the router radix to 40 (commodity hardware available in the market at the time of publication). We fix the router radix because it is typically constrained by the technology available at the time of designing the machine. We assume that each system uses the same type of nodes, routers and links. Below, we describe five *iso-X* configurations, and provide practical motivations for them. *X* represents the quantities or parameters (nodes, routers and links) kept constant across the different network topologies.

Iso-nodes (Iso-N): It is common for network designers to be provided with a target system size in terms of the number of nodes, with flexibility to choose the number of routers and links. This is because researchers and customers are often interested in comparing different systems with the same peak floating point performance (which essentially translates to having the same number of nodes). Hence, in this configuration, we fix the number of nodes at 12,000 – a number we expect to see in systems leading up to exascale. With a 50 Tflop/s node, a 12,000-node system will have a peak performance of 600 Pflop/s. With even more powerful nodes, the same system could easily provide an Exaflop/s of peak performance. We call this configuration iso-nodes, and since the number of routers and links can be chosen for each network topology independently, we try to balance the injection versus global system bandwidth.

Using a 40-port network router, we design four systems, each using one of the four network topologies under consideration – A2A dragonfly, RC dragonfly, express mesh or fat-tree (rows 1–4 in Table 2). To ensure balance, an A2A dragonfly built using 40-port routers uses 10 ports for compute nodes, 20 ports for local links,

Table 2: Design space of iso-{*} configurations: twenty systems are designed/configured in total, four each for the five *iso-X* configurations. Note: #N/#R refers to the number of nodes per router.

Configuration	Network Topology	#Nodes	#Routers	#Links	#N/#R	Balance	Other information
Iso-nodes (Iso-N)	A2A Dragonfly	12,000	1,200	17,400	10	0.97	60 groups w/ 20 routers each
	RC Dragonfly	12,250	1,750	28,875	7	0.94	35 groups w/ 10×5 routers each
	Express mesh	12,000	1,200	17,400	10	0.97	$12 \times 10 \times 10$ (gap=1)
	Fat-tree	12,000	1,500	24,000	20	1.0	3-level fat-tree, 30 pods
Iso-nodes & routers (Iso-NR)	A2A Dragonfly	12,000	1,200	15,000	10	0.83	60 groups w/ 20 routers each
	RC Dragonfly	12,000	1,200	16,200	10	0.54	30 groups w/ 10×4 routers each
	Express mesh	12,000	1,200	15,000	10	0.42	$12 \times 10 \times 10$ (gap=2)
	Fat-tree	12,012	1,100	16,000	24	0.67	3-level tapered fat-tree, 25 pods
Iso-nodes & links (Iso-NL)	A2A Dragonfly	12,000	1,200	18,000	10	1.0	60 groups w/ 20 routers each
	RC Dragonfly	12,096	1,512	18,144	8	0.6	42 groups w/ 6×6 routers each
	Express mesh	12,000	1,200	17,400	10	0.97	$12 \times 10 \times 10$ (gap=1)
	Fat-tree	12,012	1,100	16,000	24	0.67	3-level tapered fat-tree, 25 pods
Iso-nodes, routers & links (Iso-NRL)	A2A Dragonfly	12,000	1,500	24,000	8	1.33	75 groups w/ 20 routers each
	RC Dragonfly	12,000	1,500	24,000	8	0.8	30 groups w/ 10×5 routers each
	Express mesh	12,000	1,500	24,000	8	1.33	$15 \times 10 \times 10$ (gap=1)
	Fat-tree	12,000	1,500	24,000	20	1.0	3-level fat-tree, 30 pods
Iso-routers & links (Iso-RL)	A2A Dragonfly	13,500	1,500	22,500	9	1.11	75 groups w/ 20 routers each
	RC Dragonfly	10,500	1,500	24,000	7	0.91	30 groups w/ 10×5 routers each
	Express mesh	12,000	1,500	24,000	8	0.67	$15 \times 10 \times 10$ (gap=2)
	Fat-tree	12,000	1,500	24,000	24	1.0	3-level fat-tree, 30 pods

and 10 ports for global links. Accordingly, we choose the number of routers in a group to be 20 which leads to 60 groups to support 12,000 nodes. Since the system is smaller than the maximum system size possible, all 10 global links are used to provide 3 global links between a pair of groups. The construction of the RC dragonfly is similar. Seven ports are used for compute nodes which requires many more routers in this case – $12000/7 \approx 1715$. Since there are 26 ports dedicated to local links, we put $10 \times 5 = 50$ routers in each group with 2 local links per router-pair in the rows and columns. Accordingly, we choose an integer number of groups = 35, and recalculate the total number of routers and links to be 1,750 and 12,250 respectively. Because of the high router count, the RC dragonfly has the highest number of links.

In the case of express mesh, for balance, we assign 10 compute nodes to each router and the remaining 30 are used for the three dimensions. We create a $12 \times 10 \times 10$ mesh and provide $g = 1$ connections since we have enough network ports. Express mesh and A2A dragonfly require the smallest number of routers and links for building this system. A full bisection, three-level fat-tree built using 40-port routers can have 40 pods (800 routers at the leaf level) and 16,000 nodes. Since our system size is only 12,000 nodes, we need 30 pods and 600 routers at the leaf level. Each router has 20 nodes attached to it. That leads to a total of $600 + 600 + 300 = 1,500$ routers. We do not count the links connecting routers to compute nodes, which gives a total link count of $600 \times 40 = 24,000$ links.

In the next four *iso-X* configurations, we put additional constraints on the design by keeping the number of routers or links or both constant in addition to the number of nodes. In the last configuration, we keep the number of routers and links constant but not nodes. The construction of systems in each configuration is similar to the details provided above for Iso-N and we omit detailed descriptions of the construction. The summary of the design space is provided in Table 2. The balance column in the table shows how far does each configuration deviate from the balanced criterion. *Balance* (diameter-normalized links per node) is calculated as follows:

$$\text{balance} = \frac{N_{links} \times 2}{N_{nodes} \times d}$$

where N_{links} is the total number of bi-directional links (multiplied by two to obtain the number of uni-directional links), N_{nodes} is the total number of nodes, and d is the network diameter.

Iso-nodes & routers (Iso-NR): Keeping the number of nodes and routers constant allows us to do a fair comparison of network designs with the same number of nodes per router. In addition, by not putting links on all available ports on each router, we can study the impact of the total number of links on system performance, and the potential monetary savings. We fix the number of nodes at 12,000 and that of routers at 1,200 (each network has 10 nodes/router, rows 5–8 in Table 2). In this configuration, for the A2A dragonfly we only use two global links per group pair instead of three, and in the case of RC dragonfly, four global links instead of 10. Compared to the Iso-N system, the RC dragonfly system in this case has fewer groups of smaller size (10×4). With cost savings as a motivation, we build a $12 \times 10 \times 10$ mesh with gap=2 and to satisfy the router constraint, we build a 25-pod 3:2 leaf tapered fat-tree. By construction, the fat-tree system only has 1100 nodes.

Iso-nodes & links (Iso-NL): Keeping the total number of links constant across the different systems can allow us to compare performance in the case of constant global network bandwidth. In this case, each network can use a different number of routers depending on its balance criteria. We fix the number of nodes at 12,000 and number of links around 18,000 (rows 9–12 in Table 2). A2A dragonfly and express mesh have 10 nodes per router. The RC dragonfly has 8 nodes/router and the fat-tree has 24 nodes/router to stay within the link constraint. In order to satisfy the link constraint, the RC dragonfly is constructed with 42 groups of size 6×6 and can only have three global links per group pair. The express mesh is organized as a $12 \times 10 \times 10$ mesh and the fat-tree is a 25-pod 3:2 leaf tapered system.

Iso-nodes, routers & links (Iso-NRL): Often times, customers or network designers work under a budget constraint. Iso-monetary cost analysis is difficult due to fluctuating market prices and due to the potential differences between real market prices and list prices provided to customers. Hence, we do not attempt to derive the number of nodes, routers and links based on a fixed budget amount but fix the number of all these network components across the different networks. In this case, we start with 12,000 nodes and 1,500 routers (these are the smallest numbers needed to build a full bisection fat-tree); we can easily build the other three systems well within their respective limits of nodes/router. All three networks other than fat-tree have 8 nodes/router and all the systems have 24,000 links (rows 13–16 in Table 2).

Iso-routers & links (Iso-RL): Finally, we also look at a configuration in which the number of routers and links is kept constant and each network can have the maximum number of nodes it can support with the remaining ports. We fix the number of routers at 1,500 and links at 24,000 (rows 17–20 in Table 2). In this case, each network has a different number of nodes/router (see Table 2) which results in a different number of nodes in each system. Express mesh and fat-tree have 12,000 nodes whereas A2A and RC dragonfly have 13,500 and 10,500 nodes respectively.

The following network parameters are kept constant across all configurations and designs: router radix, bandwidth of each link, injection bandwidth, packet size, virtual channel size, and several overheads such as router delay, NIC delay, RDMA delay etc. The values of these parameters are provided in Section 4.

4 SIMULATION SOFTWARE AND SETUP

We briefly describe our simulation suite, network parameters that are kept constant across different network models, and the proxy applications used for generating traces for the simulations.

4.1 Network Simulation Suite

Our simulation framework is comprised of the ROSS parallel discrete-event simulation (PDES) engine, CODES simulation suite and TraceR trace replay tool. We briefly discuss the capabilities provided by each of these components of the simulation framework.

PDES in ROSS: ROSS (Rensselaer Optimistic Simulation System) [7] provides the parallel discrete-event simulation capability that drives network models in CODES. ROSS simulations comprise of logical processes (LPs) that represent individual components of the system being modeled. For example, a router is an LP in the network model. ROSS uses optimistic PDES, which enables faster simulations than the traditional conservative event processing protocols [27].

Network models in CODES: The CODES simulation suite is intended to accelerate the HPC interconnect system co-design by providing detailed, packet-level models of popular high-performance interconnects such as fat-tree [25], dragonfly [24], express mesh [19], torus [26] and other networks. The dragonfly network model can simulate both the A2A and RC dragonfly configurations. The fat tree network model supports multiple network planes and tapering of the network as options. The express mesh network model provides different gap configurations. All network models use a credit-based flow control scheme in which the upstream routers track the number of packets sent to the downstream routers. Once a packet leaves the downstream router, it sends a credit to the upstream router asking it to send another packet.

Trace replay in TraceR: TraceR simulates the execution of applications by replaying their control and communication flow on top of CODES [4]. Execution traces are used as input to simulate the control flow of an application. Traces generated using Score-P [3] in the Open Trace Format (OTF) [1, 14] and using Adaptive MPI [18] in the BigSim trace format [33] are supported.

TraceR emulates the functionality provided by MPI implementations when simulating messages in an execution trace. Point-to-point messages are simulated using either the *eager* or *rendezvous* protocol [10]. Collective operations are simulated using different algorithms based on the size of messages and communicators [31]. CODES is used to simulate the flow of traffic initiated by TraceR on the network. For multi-job workloads, the network is shared among concurrently simulated jobs and a user-defined job placement is used to assign nodes to the jobs.

4.2 Fixed Network Parameters

We keep values of the following parameters constant across all configurations and designs: bandwidth of each link (25 GB/s), injection bandwidth (25 GB/s), packet size (4,096 bytes), virtual channel buffer size (65,536 bytes) and several overheads such as router delay, NIC delay, RDMA delay etc. Link bandwidth of 25 GB/s is chosen because it corresponds to Infiniband HDR cables and the injection bandwidth is fixed at the same value to match it. We simulate one MPI process per node – the communication volume in the generated traces is assumed to represent the entire off-node communication volume irrespective of the number of MPI processes versus threads. We use the following network-specific routing schemes for the respective networks: UGAL (dragonfly), shortest-path adaptive (express mesh), and static ftree (fat-tree). These routing schemes were found to perform best for the respective networks in previous studies [20, 29].

4.3 HPC Workloads

We use six proxy applications that mimic commonly occurring communication patterns in parallel simulation codes from different science domains. Each proxy application is described below and summarized in Table 3.

Table 3: Details of the proxy applications used for generating execution traces for experiments in this paper.

Proxy app.	No. of neighbors	Msg. size (MB)
Pairs	1	4
Spread	~23	2
UMesh	~9	2
MILC-comm	8	2
pF3D-comm	20, 10	2, 1
Qbox-comm	100, 5–40	1, 8, 2

Pairs: In this proxy application, each process is paired with exactly one other process with which it exchanges one 4 MB message. The pairing of processes is done arbitrarily. This pattern is based on the communication in a matrix transpose operation, which is commonly used in linear algebra libraries.

Spread: This proxy application generalizes Pairs by assigning several (16 to 32) communicating partners to every process. Each process communicates with all of these partners by exchanging 2 MB messages. Partners for each process are selected arbitrarily.

UMesh: In UMesh, each process exchanges 2 MB messages with a carefully selected set of 6-20 partners. The partner processes for a given process are "nearby" in MPI rank space. This communication is representative of an unstructured mesh computation. In applications that perform computations over unstructured meshes, each MPI process has an arbitrary but small number of neighbors that are assigned neighboring sub-domains of the mesh.

MILC-comm: MILC-comm arranges MPI processes in a four-dimensional (4D) Cartesian grid. Each process communicates 2 MB messages with its eight immediate neighbors in the grid. This is representative of the communication pattern in applications that use structured grids, e.g. MILC [8]. MILC is used for studying quantum chromodynamics and in one of its executables, `su3_rmd`, messages are exchanged with nearest neighbors in a 4D grid of MPI processes.

pF3D-comm: MPI processes are arranged in a three-dimensional Cartesian grid of size $20 \times 10 \times n$ in pF3D-comm. Value of n is determined based on the total number of processes. `MPI_Alltoall` operations of message sizes 2 MB and 1 MB per pair are performed consecutively in sub-communicators defined along the first two dimensions as is done in pF3D [30]. pF3D is a laser-plasma interaction code used at the National Ignition Facility at LLNL. In one of its phases (wave propagation), 2D FFTs in each plane are broken down into 1D FFTs along the X and Y directions.

Qbox-comm: Three collective operations are performed consecutively in Qbox-comm: `MPI_Alltoallv`, `MPI_Bcast`, and `MPI_Allreduce`. MPI processes are arranged in a two-dimensional Cartesian grid of size $100 \times n$. All-to-all operations of message size 1 MB per pair are performed in a sub-communicator defined along the first dimension. The other two operations are invoked in the sub-communicators defined along the second dimension. This proxy application represents the communications patterns in Qbox [17], a quantum chemistry code.

Each application is run on 500, 1,000, 2,000 and 4,000 MPI processes, and execution traces in OTF2 format for each application are generated using Score-P [3]. These traces are then used to create two kinds of workloads for 12,000 nodes – one consisting only of small-sized jobs (500–2,000 MPI processes, referred to as *Small jobs*) and another with jobs of all sizes (500–4,000 MPI processes, referred to as *All jobs*). The workloads are generated by randomly selecting jobs of different proxy applications and sizes from the available pool. For each kind of workload (*Small jobs* and *All jobs*), we generate three sets – Set 1, Set 2, and Set 3, which contain different combinations of jobs. Within each set, we run two different placements of the jobs on each system – linear and random. In the linear placement, nodes are assigned to jobs in a blocked/linear fashion. In the random placement, nodes are randomly assigned to jobs. As a result, each job might be scattered over the entire system.

The same randomly generated workloads are simulated for all designs within each *iso-X* configuration and even across iso-{*} configurations other than Iso-RL. In the case of Iso-RL, each network may have a different number of nodes and the randomly generated workloads are tweaked by hand to add or subtract jobs to match the node count of the system under consideration.

5 EXPLORING THE DESIGN SPACE

We now compare the performance of the four network topology designs within each *iso-X* configuration using the simulation results. Each simulation outputs the predicted execution time of each job in the multi-job workload. We use the total communication volume, V_j, of job j, and its predicted execution time, T_j, to define a communication rate for the entire workload:

$$\text{comm. rate} = \sum_j \frac{V_j}{T_j}$$

This gives us an indication of the effective bandwidth achieved for a given workload in a particular placement and network design. The simulation also outputs the bytes sent on each network port, which we use to calculate the average and maximum traffic (load) per link. These details about the network traffic can help us in understanding the performance prediction results.

As described in Section 4, within each *iso-X* configuration, for every network topology, we ran three different sets of *Small jobs* and *All jobs* multi-job workloads under two placements – linear and random. This amounts to 48 simulations within each *iso-X* configuration, and 240 simulations in total. Figures 2–6 present the results of all these simulations.

Iso-nodes (Iso-N): In this configuration, each network design may have different number of routers and links depending on its balanced criterion. This is the typical setup used in most previous work for comparing networks. In this case, each topology can make the best use of its links and routers to provide balanced injection-to-global network bandwidth. For a fixed node count, the fat-tree network has the highest bisection bandwidth of all the networks compared in the paper. This is reflected in the communication rate achieved by the fat-tree system for the linear placement (Figure 2). The A2A dragonfly system delivers the second highest communication rate in the case of linear placement. Performance of the fat-tree and A2A dragonfly systems drops in the case of random placement

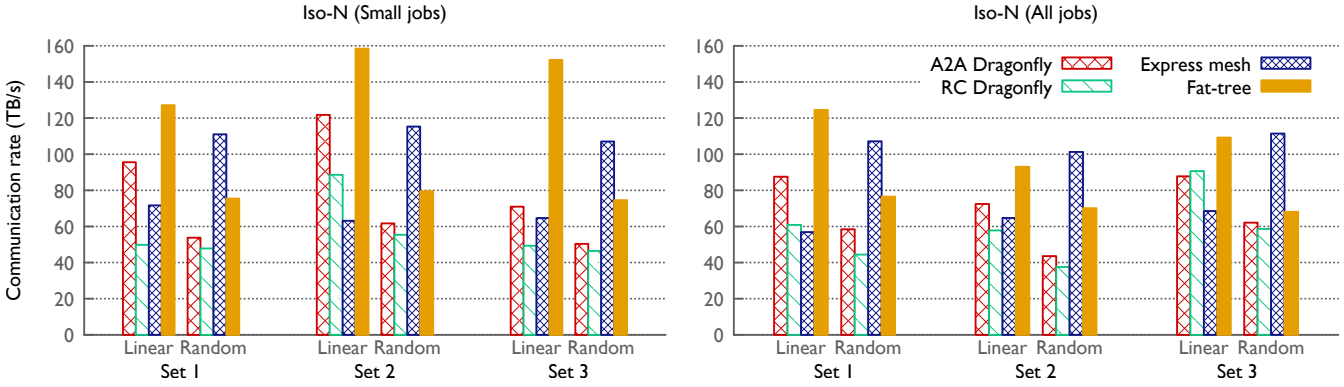

Figure 2: Performance comparison of iso-nodes (Iso-N) configurations: In the linear placement, the fat-tree topology has the highest predicted communication rate followed by the A2A dragonfly. In the random placement, express mesh has the highest predicted communication rate followed by the fat-tree topology.

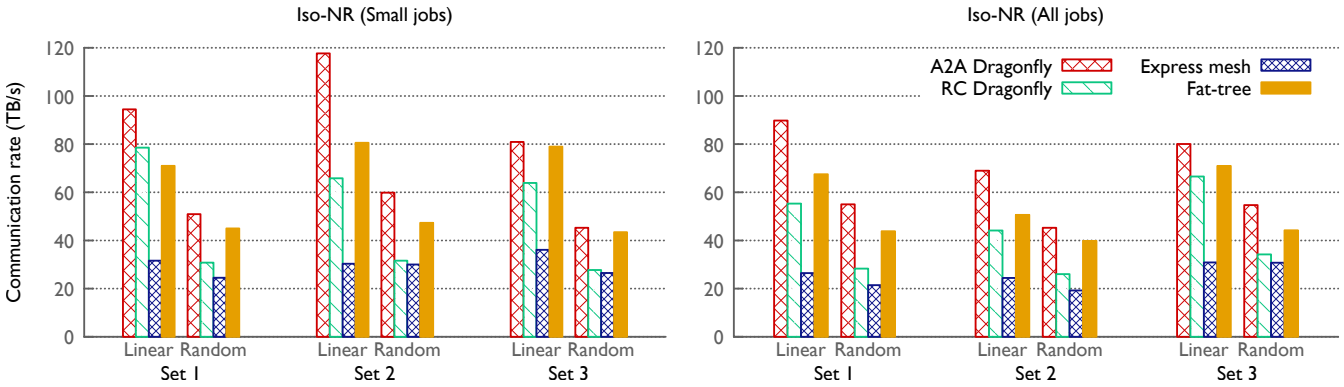

Figure 3: Performance comparison of iso-nodes & routers (Iso-NR) configurations: In this case, for both placements, the A2A dragonfly has the highest predicted communication rate followed by the fat-tree topology.

due to increased inter-job interference. This is confirmed by the 76% increase in the average and 16% increase in maximum traffic per link in the fat-tree system for the random placement (Set 2). The express mesh system is the clear winner in the case of random placement due to its performance improvements which result from a significant drop in the maximum traffic over all links.

Iso-nodes & routers (Iso-NR): In this configuration, we fix the number of routers in addition to the number of nodes, which reduces the number of links on all networks compared to the respective Iso-N configurations. This can help us evaluate the impact of cost-saving measures such as using fewer links. The communication rate drops for all networks compared to the respective Iso-N configurations as expected (Figure 3). Tapering the fat-tree and changing the gap on the express mesh network has a more adverse effect than reducing the number of global links on the dragonfly networks. As a result, the performance of fat-tree and express mesh systems drops significantly compared to their respective Iso-N designs. The performance of the A2A dragonfly does not change significantly as compared to the Iso-N design, and it emerges as the best performing one in Iso-NR configuration (for both placements). This suggests that when procuring machines, customers have the flexibility to reduce the number of optical links without noticing a huge impact on performance. The smaller group size and fewer number of groups in the case of RC dragonfly as compared to the Iso-N design leads to better performance for some workloads even with fewer global links. The fat-tree system delivers the second best performance for most workloads in both placements.

Iso-nodes & links (Iso-NL): In this configuration, the A2A dragonfly and express mesh systems are able to use all their network ports to connect routers via links. However, the fat-tree system is 3:2 leaf tapered as in the Iso-NR configuration, and the RC dragonfly has only three global links per group pair. As in the previous configuration, tapering impacts the predicted performance of the fat-tree system significantly and it is able to performs only as well as the express mesh system in the case of linear placement (Figure 4). The A2A dragonfly outperforms all other network topologies except in the case of Small Jobs' Set 3 where fat-tree is slightly better. It turns out that for some workloads in this configuration, fat-tree

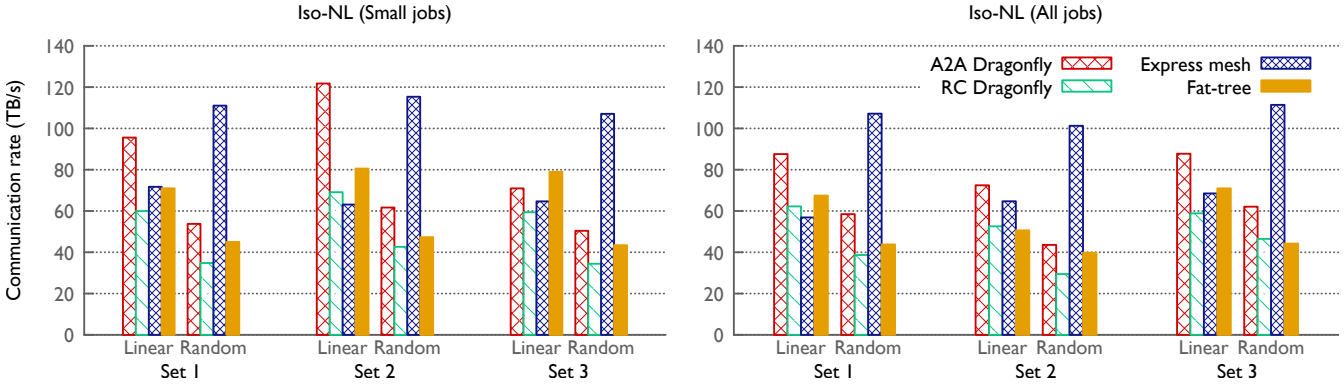

Figure 4: Performance comparison of iso-nodes & links (Iso-NL) configurations: In the linear placement, the A2A dragonfly has the highest predicted communication rate in most cases except Set 3 where fat-tree is slightly better. In the random placement, express mesh has the highest predicted communication rate followed by the A2A dragonfly topology.

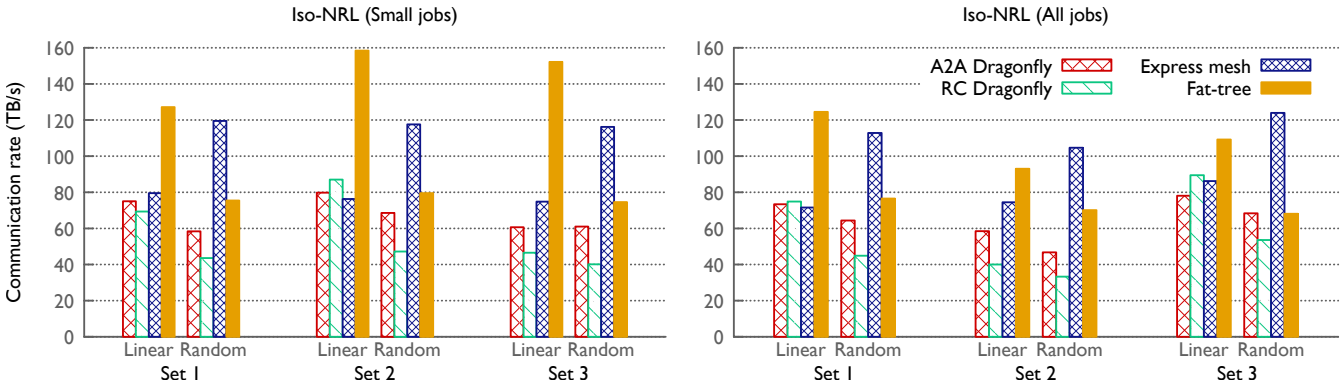

Figure 5: Performance comparison of iso-nodes, routers & links (Iso-NRL) configurations: In the linear placement, fat-tree has the highest predicted communication rate. In the random placement, express mesh has the highest predicted communication rate followed by the fat-tree topology in most cases.

has a lower average and maximum traffic per link as compared to the A2A dragonfly design. Similar to the Iso-N configuration, in the case of random placement, express mesh again outperforms other networks significantly.

Iso-nodes, routers & links (Iso-NRL): This configuration provides a fair comparison between all four network topologies since the number of nodes, routers and links is fixed. This configuration is also interesting due to the fact that all system designs have the same number of nodes per router. The results in this case, shown in Figure 5, are very similar to those in the Iso-N configuration (balanced criterion, see Figure 2). Referring back to Table 2, this suggests that in the case of express mesh and A2A dragonfly, we can build the systems with fewer routers without compromising performance significantly. As was observed in the Iso-N case, in this configuration also, the fat-tree system performs the best for linear placement, whereas the express mesh system is the best in the case of random placement followed by fat-tree.

Iso-routers & links (Iso-RL): This configuration is different from the rest because it allows each network design to have a different number of nodes. Fat-tree and express mesh have 12,000 nodes each, RC dragonfly has 10,500 nodes and A2A dragonfly has 13,500 nodes. In this case, the same workloads as in the previous iso-{*} configurations are run on the fat-tree and express mesh systems. However, in the case of RC dragonfly, we remove some jobs to fit within 10,500 nodes and in the case of A2A dragonfly, we add some jobs to fill the entire machine. Hence, we have to be careful when we compare the communication rates for this configuration since the rate is now a summation over different number of jobs for each network. It may not make sense to compare this configuration with the other iso-{*} configurations. We observe that since the number of routers and links are fixed for all networks, the results in this configuration (Figure 6) are also similar to the Iso-N and Iso-NRL configurations (Figures 2 and 5 respectively).

We summarize the predicted performance of different iso-{*} configurations except Iso-RL for the two placements (linear and

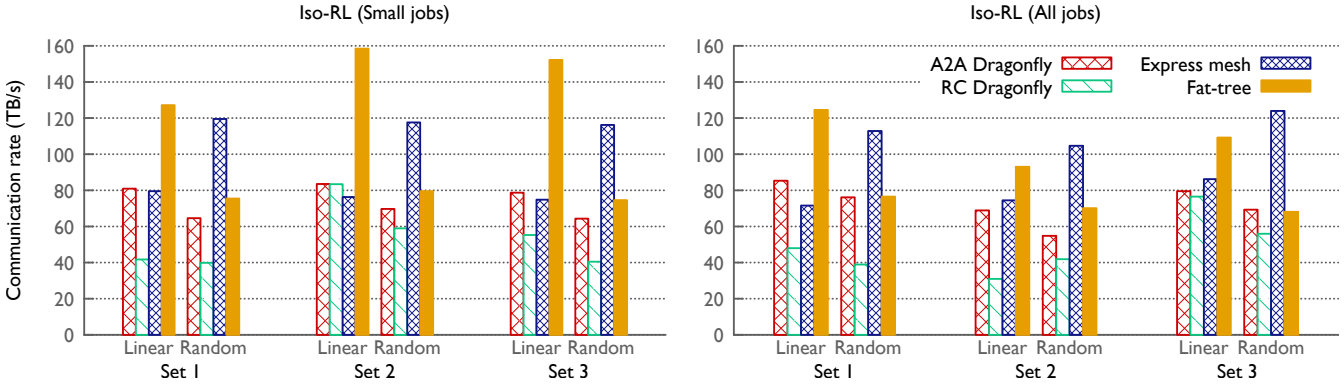

Figure 6: Performance comparison of iso-routers & links (Iso-RL) configurations: Fat-tree has the highest predicted communication rate for linear placement, and express mesh has the highest predicted communication rate for random placement.

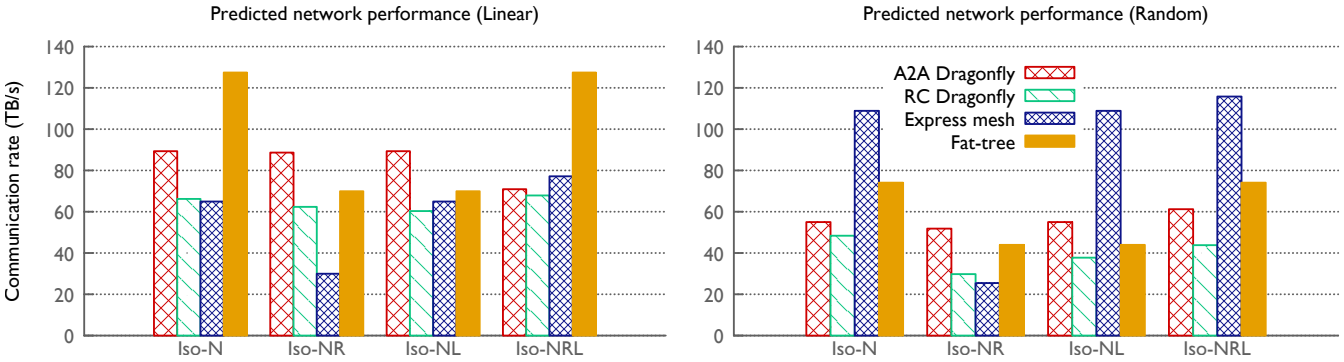

Figure 7: Predicted network performance (TB/s) of different network designs under different iso-{*} constraints. The communication rate shown is the average rate for each network design across all six workloads sets. Linear placement is shown in the left plot and random placement in the right.

random) in Figure 7. The numbers are obtained by calculating the average communication rate for each network design across all six sets of workloads (small and all). The high-level finding is that in absence of tapering, the fat-tree topology provides the best network performance in the case of linear placement, but suffers from congestion in the case of random placement. On the other hand, random placement benefits express mesh by allowing the use of more links and spreading the traffic in different dimensions. In the configurations where the fat-tree is tapered and express mesh uses gap=2, the A2A dragonfly topology performs the best. The RC dragonfly topology has the worst performance among all the networks compared in this paper for the multi-job workloads simulated here.

6 SLICING THE DESIGN SPACE

Purchasers of HPC systems attempt to optimize application performance and workload throughput under a budget constraint. Hence, understanding tradeoffs between monetary cost and application performance is important. In this section, we attempt to slice the design space by bringing monetary costs into the equation and comparing performance per dollar for different network designs.

Market prices of network components fluctuate and list prices provided by manufacturers to customers are not public. Hence, instead of using actual dollar costs of different network components, we look at their relative costs. Let us assume that the average cost of an optical cable of length 10–30 meters is x. We then say that the average cost of a copper cable of length 1-3 meters is $c \times x$ and the cost of a 40-port router is $r \times x$. Given the number of optical cables, copper cables and routers used in the system, N_o, N_c and N_r respectively, we can calculate the total network cost as:

$$\text{cost} = N_o \times x + N_c \times c \times x + N_r \times r \times x$$

where c and r are parameters that can be varied depending on the manufacturer, technology or the current market prices. In order to look at relative costs, we set the cost of an optical cable, $x = \$1$ and then find reasonable numbers for the parameters c and r. In this paper, we use $c = 0.16$, and $r = 20$. These numbers were obtained by looking at the current market prices for different manufacturers and consulting with people often involved in procurement decisions.

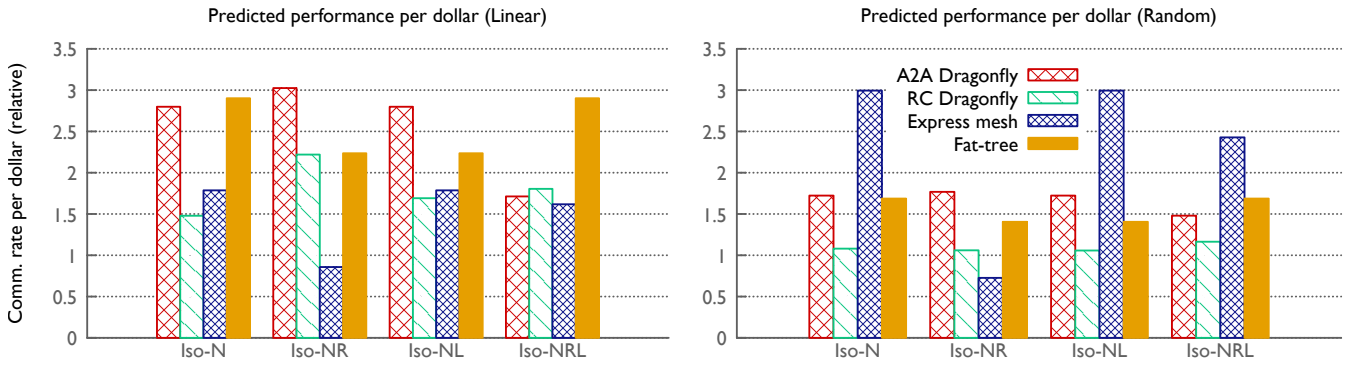

Figure 8: Understanding monetary cost versus application performance tradeoffs. The plots show the predicted network performance per dollar (GB/s/$) of different network designs under different iso-{*} constraints obtained by using relative costs of links and routers. Linear placement is shown in the left plot and random placement in the right.

Plugging in these values, the cost equation reduces to,

$$\text{cost} = N_o + 0.16 \times N_c + 20 \times N_r$$

In order to ascertain the number of links that are installed using copper cables versus optical cables, we make some assumptions for each network topology. In the case of dragonfly, we assume that all links within a group are copper cables. For fat-tree, we assume that all links within director class switches are copper cables. For express mesh, we assume that a small neighborhood around each router is connected using copper cables. Based on these assumptions and the numbers in Table 2, we derive the cost of each network (Table 4).

Table 4: Monetary costs of different network designs based on costs of copper cables and 40-port routers, calculated relative to the cost of a 10–30 meters optical cable ($1).

Config.	A2A dragonfly	RC dragonfly	express mesh	fat-tree
Iso-N	31920	44765	36360	43920
Iso-NR	29308	28104	34968	31280
Iso-NL	31920	35683	36360	31280
Iso-NRL	41400	37620	47700	43920
Iso-RL	40916	37620	47700	43920

Focusing on the Iso-NRL and Iso-RL rows first, we observe that the RC dragonfly is typically the least expensive network followed by the A2A dragonfly and fat-tree. Express mesh is the most expensive. However, by comparing Iso-NRL and Iso-N, we can make the same observation as in the previous section: we can reduce the cost of A2A dragonfly and express mesh by using fewer routers without compromising performance significantly. Finally, by using gap=2 for express mesh and tapering for fat-tree, we can bring the cost of these two networks closer to the dragonfly networks.

Using the performance summary data in Figure 7 and the cost numbers for each design in Table 4, we now calculate performance per dollar numbers for each design under different iso-{*} configurations. These are presented in Figure 8. Based on these plots, we conclude that the fat-tree and A2A dragonfly systems deliver the highest performance per dollar in the case of linear placements. Fat-tree performs better in its full bisection configuration whereas A2A dragonfly performs better when the fat-tree is tapered.

Linear job placements are often not used in practice on production supercomputers. Typically, a new job in the queue is allocated whichever nodes are currently available without any regard to the network topology. Hence, a random placement is more representative of the scenarios in practice. In this case, we notice that the express mesh system delivers the best performance among all four networks except one configuration. In the Iso-NR configuration, where we use an express mesh with gap=2, A2A dragonfly and fat-tree perform better than the express mesh system.

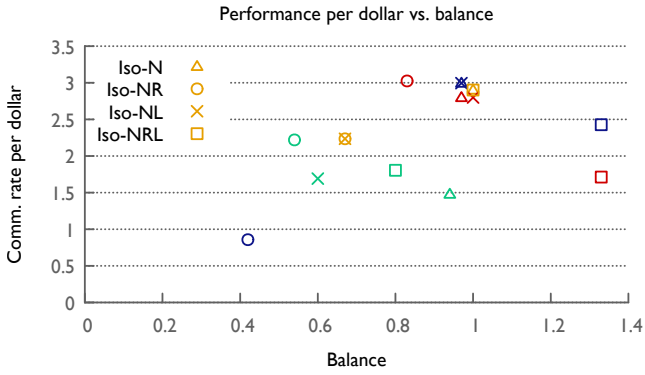

Figure 9: Predicted performance (communication rate per dollar) as a function of the *balance* property of the system (diameter-normalized links per node). The colors of the points on the plot represent different network topologies (red: A2A dragonfly, green: RC dragonfly, blue: express mesh, gold: fat-tree).

Finally in Figure 9, we summarize the trends of the predicted performance per dollar of most system configurations simulated in

this paper as a function of how balanced the networks are (diameter-normalized links per node). Each point represents a different configuration from Table 2 (Iso-RL configurations are omitted from this plot). The point shapes represent different iso-{*} configurations and the colors signify different network topologies. We can make several important observations from this plot. First, having a surplus of links in the network does not lead to the best performance per dollar (points on the extreme right). The best performance per dollar is obtained around balance = 1. However, several configurations that are not balanced (between 0.8 and 1) have better performance per dollar. They are systems built using the A2A dragonfly network or the express mesh network. It is also evident that RC dragonfly systems do not lead to high performance per dollar irrespective of the balance criterion.

7 RELATED WORK

There are several previous studies that focus on comparing HPC networks by using performance, cost, energy consumption and network properties as metrics. Publications that propose new network designs often compare the new topology with existing network topologies with respect to several metrics. In [23, 24], the authors compare the dragonfly topology against flattened butterfly, folded Clos and 3D torus networks. The metrics used are cost, cable length and network diameter.

Pan et al. [28] propose the Firefly network topology, which consists of clusters of compute nodes with intra-group communication performed through electrical cables, and inter-group communication performed using nanophotonics. They compare the proposed firefly network topology to a concentrated mesh, an optical crossbar and a dragonfly topology. The evaluation of various topologies is performed by extending the Booksim cycle accurate simulator [12, 21] with single job executions of synthetic traffic patterns and MineBench benchmarks. The comparison focuses on energy consumption of various network topologies, and the energy delay product (EDP) is used to compare the alternative topologies.

Besta et al. [9] propose the Slim Fly network topology and compare its performance against alternative topologies such as flattened butterfly, fat-tree, torus, hypercube and dragonfly networks. The radix of dragonfly, fat-tree and flattened butterfly network topologies is adjusted to give full global bandwidth and a balanced configuration. Low-radix network topologies such as torus, hypercube and long-hop use a radix of one. Single job execution is simulated by using synthetic traffic patterns such as uniform random, bit permutation and worst-case traffic. They use the Booksim cycle accurate simulator for the evaluation.

Fujiwara et al. [16] propose the Skywalk topology, which uses randomness to achieve low latency while aiming to reduce cable lengths. This work uses graph analysis and discrete-event simulations to compare the new design against hypercube, dragonfly, HyperX, and fully random topologies. The metrics used for comparison are latency, cable length and throughput.

More recently, researchers have performed comparative evaluations of popular interconnect topologies with respect to several metrics. In [20], the authors perform a comparison of the torus, fat-tree and dragonfly network topologies by using the TraceR-CODES packet-level simulation framework on network sizes with up to 730K endpoints. The performance comparison is done by simulating multi-job workloads coming from four benchmarks and two proxy applications that are representative of a variety of HPC communication patterns.

In [22], the authors perform a comprehensive evaluation of ten high-performance and data center network topologies: BCube, DCell, dragonfly, fat-tree, flattened butterfly, hypercube, HyperX, jellyfish, Long Hop and Slim Fly. They compare the throughput of different topologies using a variety of synthetic and empirically gathered traffic matrices, and by generating near worst-case traffic patterns for each topology. Chen et al. [11] also compare network topologies (fat-tree, dragonfly, dragonfly+, HyperX, Slim Fly) using synthetic patterns and proxy applications. The metrics used for evaluation are throughput, bandwidth per node and time.

Most previous work described above assume a balanced injection-to-global bandwidth constraint and then evaluate systems for that particular scenario. We identify different constraints that are typically applied when designing networks in practice, and describe these constraints in terms of the choices a network designer would make for the number of nodes, routers and links in the system. To the best of our knowledge, this is the first study that undertakes such a comprehensive evaluation of network topologies across many practically relevant iso-{*} scenarios.

8 SUMMARY

HPC system design in general and network design in particular is challenging due to multiple, sometimes conflicting constraints imposed on the design. Network designers often look for elegant solutions to network design challenges such as balancing the injection-to-global network bandwidth. However, customers might have to adhere to other constraints such as monetary costs or peak performance, while still trying to maximize performance of applications running on the system or improve overall system throughput.

In this paper, we described three important axes which define a design space for HPC systems – number of nodes, routers and links. We can vary each one independently or apply constraints on one or more quantities when designing an HPC system. We looked at five configurations that keep nodes, routers and/or links constant across the different network topologies (iso-{*} configurations). For these configurations, we compared four scalable network topologies using discrete-event simulations of relevant HPC workloads. We observed that full-bisection fat-tree systems are the best performers when jobs are placed linearly. However, when comparing against tapered fat-trees, A2A dragonfly performs better. When jobs are placed randomly, express mesh with gap=1 (i.e. HyperX) performs best. When using an express mesh with gap=2, A2A dragonfly performs better.

We also compared performance per dollar for the different networks by using relative costs of routers and links, derived from market data. We see similar trends in results for performance per dollar as in the performance results. Since dragonfly systems are less expensive, performance per dollar for the A2A dragonfly comes close to that of the fat-tree when linear placement is used. When comparing performance per dollar against how balanced the systems are, we observe two things – a perfectly balanced system or even surplus network bandwidth does not guarantee the best

performance per dollar. Second, we can sacrifice perfect balance and achieve better performance per dollar for some A2A dragonfly and express mesh systems.

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-772399). The CODES/ROSS simulation suite has been supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC02-06CH11357.

This document was prepared as an account of work sponsored by an agency of the U.S. government. Neither the U.S. government nor Lawrence Livermore National Security, LLC (LLNS), nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the U.S. government or LLNS. The views and opinions of authors expressed herein do not necessarily state or reflect those of the U.S. government or LLNS, and shall not be used for advertising or product endorsement purposes.

REFERENCES

[1] [n. d.]. Open Trace Format 2. https://silc.zih.tu-dresden.de/otf2-current/index.html.

[2] [n. d.]. Slingshot: The Interconnect for the Exascale Era. https://www.cray.com/sites/default/files/Slingshot-The-Interconnect-for-the-Exascale-Era.pdf.

[3] 2015. Score-P User Manual. https://silc.zih.tu-dresden.de/scorep-current/pdf/scorep.pdf

[4] 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.

[5] Jung Ho Ahn, Nathan Binkert, Al Davis, Moray McLaren, and Robert S. Schreiber. 2009. HyperX: Topology, Routing, and Packaging of Efficient Large-scale Networks. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09)*. ACM, New York, NY, USA.

[6] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, Jian Li, Nan Ni, and R. Rajamony. 2010. The PERCS High-Performance Interconnect. In *18th Annual Symposium on High Performance Interconnects (HOTI)*. 75–82.

[7] 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.

[8] 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).

[9] Maciej Besta and Torsten Hoefler. 2014. Slim Fly: A Cost Effective Low-diameter Network Topology. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14)*. IEEE Press, 348–359.

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

[11] D. Chen, P. Heidelberger, C. Stunkel, Y. Sugawara, C. Minkenberg, B. Prisacari, and G. Rodriguez. 2016. An Evaluation of Network Architectures for Next Generation Supercomputers. In *7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. https://doi.org/10.1109/PMBS.2016.007

[12] William Dally and Brian Towles. 2003. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[13] William J. Dally. 1991. Express Cubes: Improving the Performance of K-ary N-cube Interconnection Networks. *IEEE Trans. Comput.* 40, 9 (Sept. 1991), 1016–1023. https://doi.org/10.1109/12.83652

[14] D. Eschweiler, M. Wagner, M. Geimer, A. Knüpfer, W. E. Nagel, and F. Wolf. 2012. Open Trace Format 2: The Next Generation of Scalable Trace Formats and Support Libraries. 22 (2012).

[15] 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.

[16] I. Fujiwara, M. Koibuchi, H. Matsutani, and H. Casanova. 2014. Skywalk: A Topology for HPC Networks with Low-Delay Switches. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. 263–272. https://doi.org/10.1109/IPDPS.2014.37

[17] 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.

[18] 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*.

[19] Nikhil Jain, Abhinav Bhatele, Xiang Ni, Todd Gamblin, and Laxmikant V. Kale. 2017. Partitioning Low-diameter Networks to Eliminate Inter-job Interference. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (IPDPS '17)*. IEEE Computer Society. LLNL-CONF-706801.

[20] 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.

[21] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis, and J. Kim. 2013. A detailed and flexible cycle-accurate Network-on-Chip simulator. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 86–96.

[22] Sangeetha Abdu Jyothi, Ankit Singla, P. Brighten Godfrey, and Alexandra Kolla. 2016. Measuring and Understanding Throughput of Network Topologies. In *Supercomputing 2016 (SC'16)*. Salt Lake City, UT.

[23] J. Kim, W. Dally, S. Scott, and D. Abts. 2009. Cost-Efficient Dragonfly Topology for Large-Scale Systems. *IEEE Micro* 29, 1 (Jan 2009), 33–40.

[24] 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.

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

[26] 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).

[27] D. M. Nicol, C. C. Michael, and P. Inouye. 1989. Efficient Aggregation of Multiple PLs in Distributed Memory Parallel Simulations. In *Proceedings of the 21st Conference on Winter Simulation (WSC '89)*. ACM, New York, NY, USA, 680–685. https://doi.org/10.1145/76738.76825

[28] Yan Pan, Prabhat Kumar, John Kim, Gokhan Memik, Yu Zhang, and Alok Choudhary. 2009. Firefly: Illuminating Future Network-on-chip with Nanophotonics. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*. ACM, New York, NY, USA, 429–440. http://doi.acm.org/10.1145/1555754.1555808

[29] Bogdan Prisacari, German Rodriguez, Philip Heidelberger, Dong Chen, Cyriel Minkenberg, and Torsten Hoefler. 2014. 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 (HPDC '14)*. ACM, 129–140. https://doi.org/10.1145/2600212.2600225

[30] 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.

[31] 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.

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

[33] Gengbin Zheng, Gunavardhan Kakulapati, and Laxmikant V. Kale. 2004. BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines. In *18th International Parallel and Distributed Processing Symposium (IPDPS)*. Santa Fe, New Mexico, 78.