Partitioning Low-diameter Networks to Eliminate Inter-job Interference

Nikhil Jain*, Abhinav Bhatele*, Xiang Ni†, Todd Gamblin*, Laxmikant V. Kale‡

*Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, California 94551 USA
†IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598 USA
‡Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801 USA
E-mail: *{nikhil, bhatele, tgamblin}@llnl.gov, †xiang.ni@ibm.com, ‡kale@illinois.edu

***Abstract*—On most supercomputers, except some torus network based systems, resource managers allocate nodes to jobs without considering the sharing of network resources by different jobs. Such network-oblivious resource allocations result in link sharing among multiple jobs that can cause significant performance variability and performance degradation for individual jobs. In this paper, we explore low-diameter networks and corresponding node allocation policies that can eliminate inter-job interference. We propose a variation to n-dimensional mesh networks called *express mesh*. An express mesh is denser than the corresponding mesh network, has a low diameter independent of the number of routers, and is easily partitionable. We compare structural properties and performance of express mesh with other popular low-diameter networks. We present practical node allocation policies for express mesh and fat-tree networks that not only eliminate inter-job interference and performance variability, but also improve overall performance.**

***Keywords*-network topology; partitionability; inter-job interference; express mesh; simulation;**

I. Introduction

Computational power of high performance computing (HPC) systems has been increasing at a fast rate for several years. This has led to network resources becoming a major performance bottleneck when executing applications at extreme scales. Low-diameter, high-radix networks such as fat-tree (FT) [1], dragonfly (DF) [2], [3], and Slim Fly (SF) [4], are being explored to cope with the scarcity of network resources.

Most resource managers allocate nodes to jobs using network-oblivious schemes that maximize job throughout and system utilization [5]. A major side-effect of such allocation schemes is that while compute resources are dedicated to individual jobs, network resources such as routers and links are shared among multiple jobs. Further, non-minimal adaptive routing in low-diameter network (LDN) topologies such as DF and SF increases this sharing of resources and makes it harder to allocate network resources exclusively to individual jobs.

Sharing of network resources among multiple jobs increases network congestion and results in inter-job interference. Recent studies have shown that inter-job interference causes significant variation and degradation in observed performance of applications [6], [8]. Figure 1 presents one such example in which a production application is run on a 5D torus based system (Mira) and a dragonfly-based system (Edison) several times. On an LDN such as DF in which network resources are shared (not partitioned), up to 2× performance variability is observed. Performance variability makes run-to-run comparisons of different executions difficult and hampers the process of optimizing code performance. In contrast, an easily partitionable torus network provides consistent performance. In this paper, *partitionability* refers to a property of the network that facilitates network-aware allocation of nodes to jobs with a goal of minimizing link sharing among jobs or partitions.

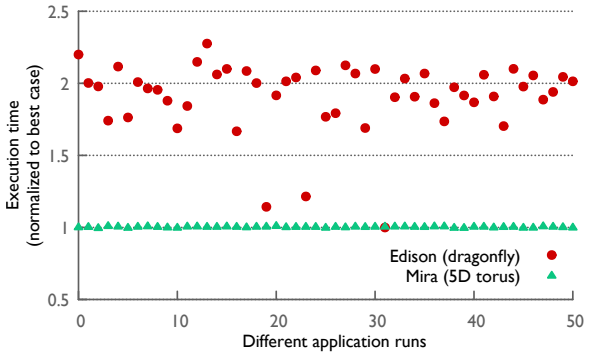


Figure 1: Run-to-run application performance variability in current HPC systems [6], [7]

We address the problem of inter-job interference by attempting to answer the following: can a combination of network topology (existing or new design) and node allocation policy eliminate inter-job interference without losing the performance achievable on shared low-diameter networks? To address this challenge, we study the partitionability of three well-known LDN topologies – DF, FT, and SF.

Mesh and torus networks result in lower performance as compared to LDNs because of their large diameter. Even when high-dimensional meshes and tori are used, the diameter increases rapidly with node count, while the bisection bandwidth does not increase as fast as that on LDNs such as DF and SF. However, mesh and torus networks can be partitioned easily to provide isolated allocations to each job. This results in predictable performance on these systems as demonstrated by the results on Mira in Figure 1. Driven by these observations, we explore variations to mesh networks that can reduce network diameter and improve performance, while retaining the ability to provide interference-free node allocations to individual jobs.

The main contributions of this paper are:

- We present *express mesh*, a generalization of mesh-based LDN topologies, and describe deadlock-free routing policies for it.
- We compare structural properties and performance of express mesh with other LDN topologies.
- We propose two new metrics to quantify the expected interference for a given node allocation policy.
- We present node allocation policies that result in interference-avoiding partitioning of express mesh and fat-tree and demonstrate performance improvements of 7–71% using them.

II. Express Mesh

N-dimensional mesh and torus networks have been used in many supercomputers in recent years. However, due to their node-count dependent large diameters, even high-dimensional meshes and tori provide lower performance in comparison to low-diameter networks. To address this limitation, we introduce a variation to n-dimensional mesh networks, called express mesh (EM), that has a low diameter independent of the number of routers and high bisection bandwidth. In Section IV-B, we show that despite its low-diameter, EM is easily partitionable.

A. Construction

Building an n-dimensional (n-D) mesh requires routers with only $2 \times n$ ports. However, several LDNs utilize cost-effective, high-radix routers available today to improve their performance. In EM, we use available ports on high-radix routers to connect routers that are several hops away in an n-D mesh. Our goal is to create a topology that limits the maximum number of hops between any pair of routers to a low value that is independent of the total number of routers. The idea of adding connections (called *express channels*) to distant routers for reducing diameter has been used previously to create express cubes [**?**] and HyperX [9]. In EM, we generalize this idea to enable design of large networks given a router radix.

The construction of EM begins with an n-D mesh as its base, of dimensions $k_0 \times k_1 \times \cdots \times k_{n-1}$, where k_i is the length of dimension i. A router in the mesh with ID, r_j, is assigned unique coordinates: $\texttt{COORD}(r_j) = \{l_0, l_1, \cdots, l_{n-1}\}$. Let RID be the inverse function of COORD, i.e. given coordinates c_r of a router, $\texttt{RID}(c_r)$ returns the router ID r_j such that $\texttt{COORD}(r_j) = c_r$. The set of all routers that a router r_j is connected to in an n-D mesh is defined as:

$$mcon(r_j) = \left\{ \bigcup_{i \in [0,n)} \texttt{RID}(\{\cdots, l_i + 1, \cdots\}) \mid l_i + 1 < k_i, \bigcup_{i \in [0,n)} \texttt{RID}(\{\cdots, l_i - 1, \cdots\}) \mid l_i - 1 \geq 0 \right\} \quad (1)$$

Eq. 1 states that each router is connected to its neighboring routers (if they exist) in both directions along each of the mesh dimensions. Next, we use additional ports to connect a router with distant routers located at arithmetically increasing distance along every mesh dimension. This requires selecting a second parameter called *gap* that determines the layout of additional connections. If the gap is g, within each dimension, we add a link between a router, r_j, and all routers whose distance in that dimension from the immediate neighboring routers of r_j in the original mesh is a multiple of g. Note that we only add links within each dimension and not between routers whose coordinates differ in more than one dimension.

Formally, the set of all routers that a router j is connected to in EM is defined as:

$$\begin{aligned} econ(r_j) = \{ & mcon(r_j), \\ & \bigcup_{i \in [0,n)} \texttt{RID}(\{\cdots, l_i + 1 + m \times g, \cdots\}) \\ & \quad \mid l_i + 1 + m \times g < k_i \wedge m \in [1, \lceil k_i/g \rceil], \quad (2) \\ & \bigcup_{i \in [0,n)} \texttt{RID}(\{\cdots, l_i - 1 - m \times g, \cdots\}) \\ & \quad \mid l_i - 1 - m \times g \geq 0 \wedge m \in [1, \lceil k_i/g \rceil] \} \end{aligned}$$

In Eq. 2, connections from a router can be divided into three groups: connections inherited from the original mesh, connections added in the negative direction, and connections added in the positive direction. Depending on the coordinates of a router, different number of connections belong to each of these groups. This connectivity guarantees that every router is reachable in $n \times g$ hops from any other router.

Figures 2(a), (b) show examples of connectivity in EM constructed using n = 1, g = 1, k_0 = 8. For clarity, the first figure shows the connections originating from only two routers. For the first router in the 1D mesh ($l_0 = 0$, shown in red), six new connections are added in the positive direction since $\forall m \in [1, 6]$, $l_0 + 1 + m$ is less than k_0. In contrast, for the router on the extreme right ($l_0 = 7$), all new connections are added in the negative direction. For all other routers, 5 new connections are added, unevenly split between the positive and negative direction depending on the routers' coordinates.

When n = 1, g = 2 are used (Figures 2(c), (d)), each router has four connections in EM. For example, router with coordinate $l_0 = 4$ (shown in green) inherits two connections from the 1D mesh: left and right neighbors ($l_0 = 3$ and $l_0 = 5$), and gets one additional connection to routers that are at gap 2 from its existing neighbors in each direction ($l_0 = 1$ and $l_0 = 7$). It can be seen that the maximum distance between any two routers is now two.

When $n > 1$, connections are added between routers within each dimension independently, in the same fashion as in a 1D EM. Figures 2(e), (f) present two such examples in which n = 2, k_0 = 8, k_1 = 6 and the gap is set to one and two, respectively. When $g = 1$, we obtain a HyperX network with all-to-all connectivity along every dimension.

Claim: The shortest path between a pair of routers in an n-D g-gap EM is at most $n \times g$ hops.

Proof: Let us examine an arbitrary pair of source router, $r_s = \{s_0, s_1, \cdots, s_{n-1}\}$ and destination router $r_d = \{d_0, d_1, \cdots, d_{n-1}\}$. The number of hops between these two routers, H, is the sum of hops traversed in each of the n

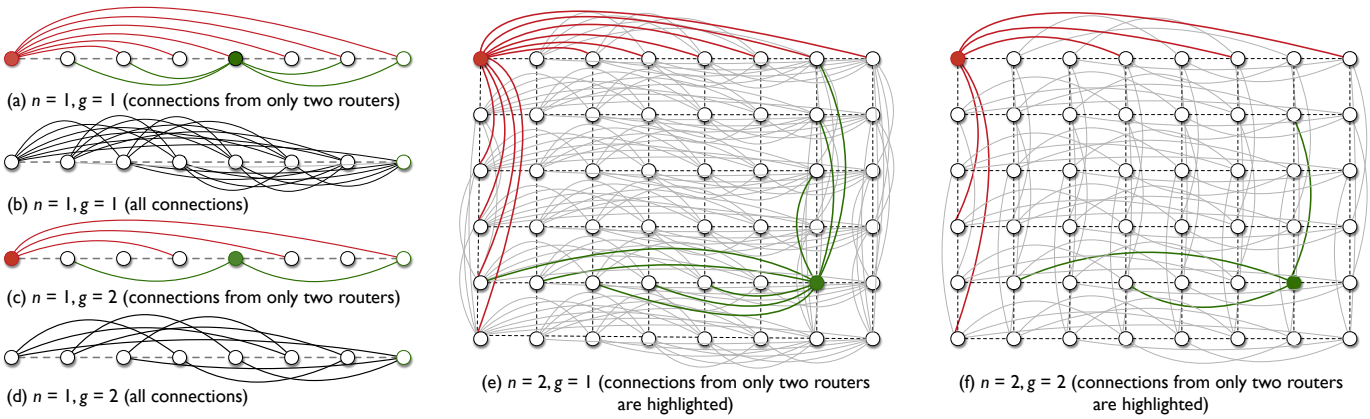


Figure 2: Construction of one- and two-dimensional express meshes with gap, $g = 1$ and 2. Connections that are inherited from n-dimensional mesh are shown as dashed straight lines; new connections added are shown as solid curved lines.

dimensions, i.e. $H = \sum_{i=0}^{n-1} h_i$. For each dimension, i, we can determine the smallest number of hops that needs to be traversed as follows:

1) $s_i = d_i$: no traversal required, thus $h_i = 0 < g$.

2) $s_i = d_i + 1$ or $s_i = d_i - 1$: direct connection inherited from the original mesh, thus $h_i = 1 \leq g$.

3) $s_i + 1 + m \times g = d_i$ or $s_i - 1 - m \times g = d_i$ for some m : direct connection available in EM, thus $h_i = 1 \leq g$.

4) other cases : if there is no direct connection to go from s_i to d_i, we can identify the closest router to s_i that has a direct connection to d_i. Since d_i must have direct connections every g routers, we should be able to find one within $g - 1$ hops from s_i. These can be near-neighbor hops inherited from the original mesh and then we make one additional hop from the identified router to d_i. Hence, $h_i \leq g$.

Since $h_i \leq g$ for any i, $H = \sum_{i=0}^{n-1} h_i \leq n \times g$.

B. Node Count for Balanced Express Mesh

We now derive the formula for the number of nodes, p, that can be connected per router in an EM in order to create a balanced system. In a balanced system (also defined in [2], [4]), all nodes can inject traffic at peak injection bandwidth for an all-to-all pattern, in which all nodes simultaneously communicate with all other nodes. The number of nodes per router, also called *concentration* [4], is equal to the number of ports reserved for injection of network traffic. For a network with router radix, r, number of nodes per router, p, network links per router, $r' = r - p$, diameter d, and average number of hops h_{avg}, these quantities are related as:

$$p \approx \frac{r'}{h_{avg}} \approx \frac{r}{(h_{avg} + 1)} \tag{3}$$

Eq. 3 holds because average number of hops also represents average load on any link in the system. Thus, to sustain peak injection bandwidth, only $\frac{r}{h_{avg}+1}$ ports can be used for injection and $\frac{r \times h_{avg}}{h_{avg}+1}$ ports are needed for communicating messages in a balanced system. For many LDN topologies, average hops is close to the network diameter, thus d can be used instead of h_{avg} in Eq. 3. For example, on SF with $d = 2$, $r/3$ ports should be used for injection [4], while on DF with $d = 3$, $r/4$ ports should be used for injection [2].

Consider an n-D g-gap EM with N_r routers arranged as $k_0 \times k_1 \times \cdots \times k_{n-1}$. In dimension i, most routers are connected to two neighboring routers (as in a mesh) and $\frac{k_i - 3}{g}$ other routers. Thus, the concentration is

$$p \approx \frac{1}{h_{avg}} \times \sum_{i=0}^{n-1} \left(2 + \frac{k_i - 3}{g}\right) \tag{4}$$

For a symmetric EM in which all dimensions are of length k, Eq. 4 reduces to $p = \frac{n}{h_{avg}} \left(2 + \frac{k-3}{g}\right)$.

When $g = 1$, the average hops is close to its diameter $n \times g$. Thus, for $n = 2$ and $n = 3$, nodes per router is $r/3$ and $r/4$, respectively. However when $g = 2$, average hops for $n = 2$ and $n = 3$ are approximately 3 and 4, respectively. Thus, we obtain $p = r/4$ for $n = 2$ and $p = r/5$ for $n = 3$.

C. Routing Schemes

In this section, we present static and adaptive routing schemes for EM and discuss mechanisms that are used to guarantee deadlock freedom in both cases. Let s be a node attached to router r_s with coordinates $\{s_0, \cdots, s_{n-1}\}$ that wants to send messages to node d on destination router r_d with coordinates $\{d_0, \cdots, d_{n-1}\}$. Messages originate at s and are sent to the attached router r_s if the destination node $d \neq s$. From r_s, using the schemes defined next, messages are forwarded to r_d and are eventually sent to d.

1) Static Routing: In this scheme, traffic is sent from source to destination in dimension-order as is common in n-D meshes. Let us choose an arbitrary order of dimensions to send all traffic in the system: $o_0, o_1, \cdots, o_{n-1}$. In dimension-order routing, a message that originates at r_s is first communicated along dimension o_0 to router r_t whose all but o_0th coordinate are the same as r_s. The o_0th coordinate of r_t is d_{o_0} – same as the o_0th coordinate of r_d. Next, the message is sent along dimension o_1 to router $r_{t'}$ such that all but o_1th coordinate of $r_{t'}$ are same as r_t. The o_1th coordinate of $r_{t'}$ is equal to the o_1th coordinate of r_d. This process is then continued

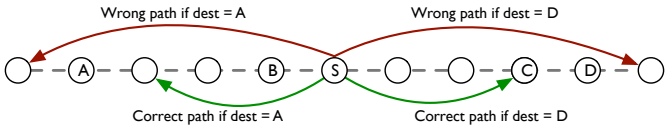


Figure 3: Routing within a dimension: messages always travel toward the destination, but do not use additional links to go beyond the destination.

through dimensions $o_2, o_3, \cdots, o_{n-1}$ until the message reaches its destination router.

When a message has to be communicated from one router, r_t with coordinates $\{t_0, \cdots, t_i, \cdots, t_{n-1}\}$, to another, $r_{t'}$ with coordinates $\{t_0, \cdots, t_i^{'}, \cdots, t_{n-1}\}$, whose coordinates differ only in the ith dimension, the following rules are used:

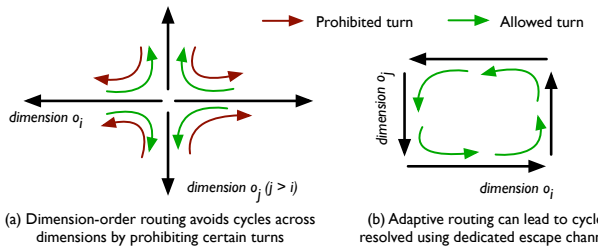
1) if r_t and $r_{t'}$ are connected by a direct link in dimension i : use the direct link to reach $r_{t'}$ from r_t. In Figure 3, examples of this case are messages sent from S to B or C.

2) if $t_i < t_i^{'}$: either forward the message to the router with ith coordinate $t_i + 1$ using the direct link, or forward the message to router $r_{t''}$ with the largest ith coordinate less than $t_i^{'}$ and with a direct connection to r_t in the ith dimension. In this case, a message is never sent to a router whose ith coordinate is greater than $t_i^{'}$. An example for this case is shown in Figure 3 as messages sent from S to D.

3) if $t_i > t_i^{'}$: either use the direct link to forward the message to the router with ith coordinate $t_i - 1$, or forward the message to router $r_{t''}$ with the smallest ith coordinate greater than $t_i^{'}$ and with a direct connection to r_t in the ith dimension. In this case, a message is never sent to a router whose ith coordinate is less than $t_i^{'}$. Messages sent from S to A fall under this category in Figure 3.

Dimension-order static routing is known to be deadlock-free for n-D meshes since it prohibits formation of cycles across dimensions as shown in Figure 4(a). Thus, for dimension-order static routing to be deadlock-free in EM, we only need to ensure that cycles cannot form within a dimension. In EM, like in an n-D mesh, a message always flows away from the source and toward the destination. While use of the new express mesh connections can cause a message to *change direction*, this is prevented by prohibiting messages from traversing farther than their destination coordinate. Since messages do not change direction and bi-directional links are available for communication in opposite directions, cycles cannot form within any dimension. Thus, dimension-order static routing is deadlock-free in EM.

2) Adaptive Routing: In this scheme, instead of following a pre-assigned order of dimensions for traversal, packets (created by breaking down messages) are forwarded toward the destination router in the dimension with least congested links. Congestion on different links connected to a router is estimated by keeping track of available tokens for all directly connected routers. Each virtual channel is initially assigned a fixed total token count (equal to the size of the virtual channel buffers). Whenever a packet is forwarded on a link, the token count for the associated virtual channel is decremented. The token



(a) Dimension-order routing avoids cycles across dimensions by prohibiting certain turns

(b) Adaptive routing can lead to cycles; resolved using dedicated escape channels

Figure 4: Deadlock freedom across dimensions: adaptive routing relies on escape channels using dimension-order routing to break deadlocks [10].

count is incremented when an *ack* or *credit* control message is received for the virtual channel from the connected router.

For both static and adaptive routing schemes, when no tokens are available for a virtual channel, new packets cannot be sent on that channel. In adaptive routing, when a packet is to be sent, token counts of all possible virtual channels that the packet can be forwarded on are compared, and the channel (and hence the corresponding link) with highest token count is chosen for forwarding the packet.

Figure 4(b) shows that adaptive routing can lead to deadlocks because cycles can form among different message flows. In EM, we resolve such deadlocks using the *escape* channel scheme proposed by Puente et al. [10]. In this scheme, in addition to channels that use adaptive routing, every link also contains an escape channel. Packets traveling in escape channels follow dimension-order routing and thus do not deadlock. When packets in adaptive channels cannot move forward but tokens are available in escape channels, packets are temporarily moved to escape channels and forwarded. Packets can be moved back to adaptive channels at any point in their path if tokens are available in adaptive channels. This ensures that the adaptive routing can resolve deadlocks as they happen.

III. Structural Analysis and Performance Comparison

We begin with a comparison of the structural properties of EM with other networks and then use simulations to compare performance of multi-job workloads. We refer the reader to the following publications for details of other networks – hypercube [11], fat-tree (FT) [1], torus [12], dragonfly (DF) [2], [13], and Slim Fly (SF) [4]. In the analyses below, topology parameters of all networks have been chosen to create balanced systems. For EM, we focus on the instances with $n = 3$ (3D EM) and $g = 1, 2$ because these parameters can be used to deploy sufficiently large systems (>100K nodes).

A. Structural Analysis

The metrics compared below describe the structure of HPC interconnection networks:

1) Diameter and Hops: Figure 5(a) shows that the diameter of 3D EM with $g = 1$ (G1 EM) is always three and is equal to the diameter of DF. This is less than the diameter of FT, but worse than SF whose diameter is two. The diameter of

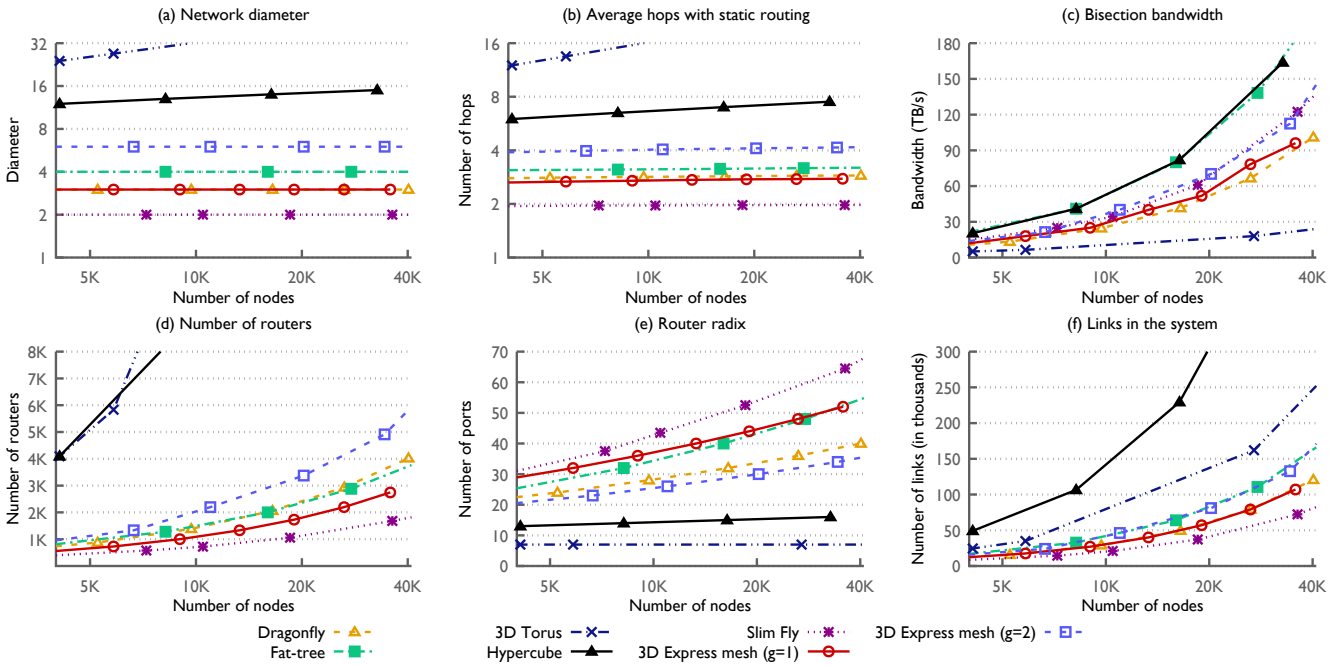


Figure 5: Structural analysis of express mesh and comparison with other networks.

3D EM with $g = 2$ (G2 EM) is twice as that of G1 EM, but as Figure 5(b) shows, the average number of hops between any two routers is much lower (four). Average hops for SF is lowest (close to two), followed by G1 EM and DF. Note that these average hops are computed assuming static shortest-path routing. While the average hops for EM and FT do not change when adaptive routing is used, it typically increases by $2\times$ for SF and DF.

2) Bisection Bandwidth: Figure 5(c) compares the bisection bandwidth of various networks assuming each link has a bandwidth of 10 GB/s. FT and hypercube provide the highest bisection bandwidth for a given node count followed by G2 EM and SF. The bisection bandwidth of G1 EM and DF is similar and is lowest among the high-radix networks.

3) Router and Link Attributes: Figures 5(d-f) present the router and link requirements of various networks for different node counts. For a fixed router radix, system sizes feasible with G2 EM and DF are larger than G1 EM, FT, and SF. For example, to build systems with >27K nodes, FT, G1 EM, and SF require routers with >50 ports. In contrast, G2 EM and DF can be used to build systems of >100K nodes using routers with 44–48 ports.

Since getting reasonable estimates of dollar costs of routers and links used in HPC systems is difficult, we use the data in Figures 5(d-f) to compare the relative costs of different networks. For a given node count, SF needs the minimum number of routers but it needs higher router radix than other networks. If we assume that the cost of routers increases linearly with their radix, the total cost of routers for SF is expected to be 30% less than all other topologies.

Router radix required for G1 EM and FT converges as the number of nodes increases. Since G1 EM requires 20–30% fewer routers, we expect the total cost of routers for G1 EM to be 20–30% less than that on FT. While the router count required for DF is typically higher than G1 EM, the router radix required by DF is proportionally lower. Thus, we expect the total cost of routers for DF and G1 EM to be similar. Finally, since G2 EM requires a significantly lower router radix than G1 EM but a higher router count than G1 EM, we expect the total cost of routers for G2 EM to be higher than G1 EM, but similar to FT.

Based on the link counts shown in Figure 5(f), the total cost of links for SF is expected to be lowest, followed by G1 EM and DF. Both FT and G2 EM require similar number of links, with counts greater than the other three networks.

B. Performance Comparison

We compare the performance of different network topologies using packet-level, discrete-event simulations of five representative communication patterns and multi-job workloads that consist of these patterns. Two of the patterns (*Unstructured Mesh* and *Stencil*) represent communication in unstructured and structured grid applications. *Pairs* and *Spread* are representative of long distance communication patterns in applications in which processes communicate with one or many processes. The last pattern (*Sub-AlltoAll*) is derived from applications that perform multi-dimensional FFTs, which result in all-to-all communication within sub-communicators.

For each topology, we construct balanced prototypes that have approximately 10.7K nodes. Details of each system are in Table I. Four processes per node are simulated on all the prototype systems. For a fair comparison, the total number of processes is fixed at 40,000 for all simulations.

TABLE I: Configurations of prototypes using different networks

System	Router radix	Nodes/ router	No. of nodes	Other information
Dragonfly	29	7	11,130	106 15-router groups
Fat-tree	44	22	10,648	3-level fat-tree
Slim Fly	44	15	10,830	722 routers
G1 EM	38	9	10,800	$12 \times 10 \times 10$ mesh
G2 EM	24	5	10,080	$14 \times 12 \times 12$ mesh

Simulations are performed using network models implemented in CODES [14], driven by TraceR [15]. The communication traces used as input for TraceR have been collected by running MPI communication kernels using Adaptive MPI [16]. Each simulation executes multiple iterations of the communication patterns and predicts the execution time for running each pattern. We compute the total amount of communication generated by each communication pattern and divide it by the predicted execution time to obtained the predicted *communication rate*. Higher communication rates signify better network performance.

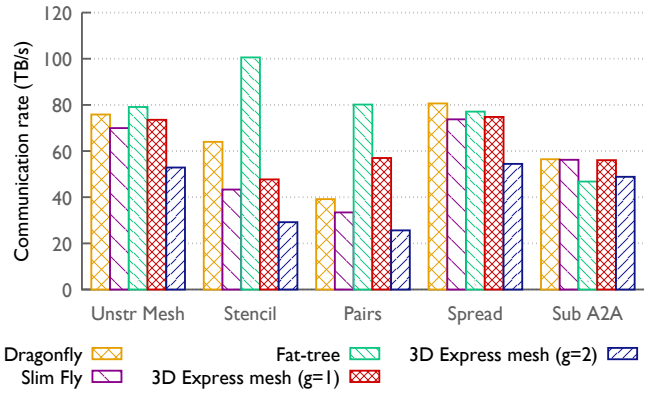


Figure 6: Performance predictions using adaptive routing and best of linear and random mapping.

1) Single Job Runs: Figure 6 presents the predicted communication rates when a single communication pattern is executed on the entire system. Results are shown only for adaptive routing because the predicted communication rates for adaptive routing are significantly better than that for static routing on all networks, except FT. On FT, performance of static routing is similar to adaptive routing for some communication patterns (e.g. Stencil), but worse for other patterns (e.g. Spread).

For Unstr Mesh and Spread, all networks exhibit similar performance with DF and FT being marginally better than SF and G1 EM. However, for Stencil and Pairs, FT has a significantly higher communication rate. For Pairs, the communication rate on G1 EM is 42% higher than DF and SF, but is 26% lower than FT. In contrast, for Sub A2A, DF, SF, and G1 EM provide 21% higher communication rates than FT. For single job runs, the communication rates on G2 EM are up to 30% worse than those on G1 EM.

2) Multi-job Runs: Next, we compare the performance of different networks for scenarios in which multiple jobs representing different communication patterns are executed simultaneously. We focus only on large jobs, each running on 5–40% of the total number of nodes. To generate a multi-job workload, we randomly pick different job sizes till we have enough jobs to occupy the entire system, and then randomly assign different communication patterns to each job. This process is repeated with different randomization seeds to generate many workloads. The node allocation policy is *clustered* – most of the nodes assigned to a job are topologically close, but ~10% may be arbitrarily scattered on the system. We use a clustered allocation to approximate the behavior of typical resource managers on production systems.

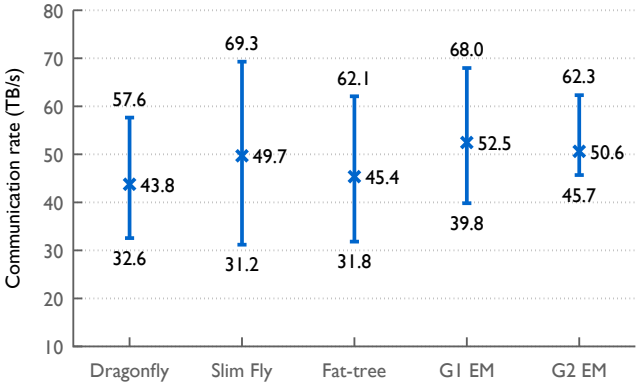


Figure 7: Comparing performance on different networks using 30 different multi-job workloads. Values shown are minimum, median, and maximum across different workloads.

The communication rate of a multi-job workload is computed by aggregating the communication rate of individual jobs in the workload. Figure 7 summarizes the results obtained for simulating 30 randomly generated workloads. The median predicted communication rate over all workloads is highest on G1 EM, closely followed by G2 EM and SF. The minimum communication rate on G1 and G2 EM is also significantly better than other networks. SF results in highest communication rate for some workloads, but it also leads to the worst performance for some workloads. The performance of DF and FT is generally lower, but the overall difference is at most 20%.

In order to understand the reasons for the obtained results, we analyzed the link traffic statistics for each network. We find that while FT has the lowest maximum load on any link among all the networks, the average traffic on links that are used is significantly higher than the overall average. This indicates transient congestion and is the most likely cause of lower performance of FT. On DF, we find that certain links are always significantly more congested than other links, and create persistent network hot-spots that degrade DF's overall performance. The traffic distributions on EM and SF were found to exhibit less significant variations across links.

IV. Partitionability

In the previous section, we saw that the predicted performance of several LDNs differs by at most 20% for multi-

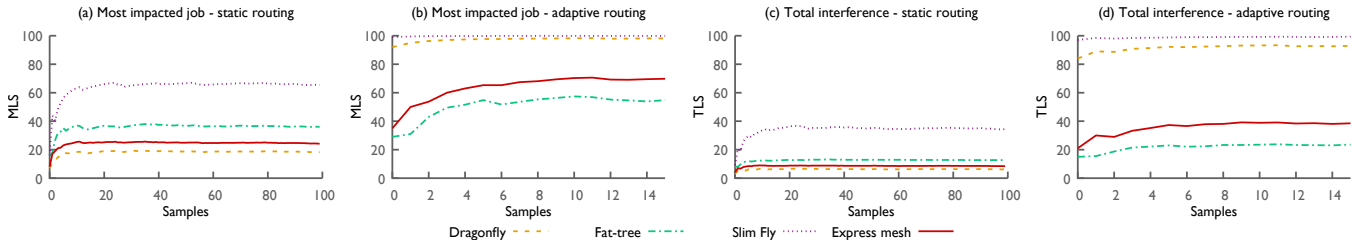


Figure 8: Moving average showing expected interference (all jobs in each workload run Spread at different node counts).

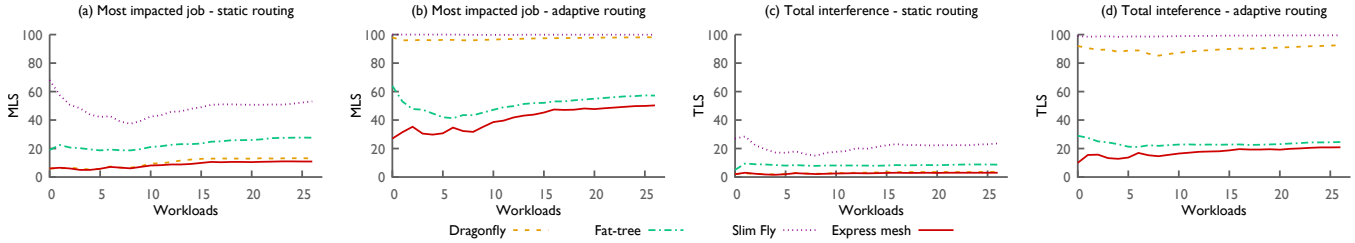


Figure 9: Moving average showing expected interference (each workload consists of different communication patterns).

job workloads. Next, we compare the expected inter-job interference on these networks for multi-job workloads and examine techniques to partition different networks in order to eliminate inter-job interference.

A. Quantifying Expected Inter-job Interference

When a multi-job workload is executing on a system, the amount of inter-job interference depends on several factors – the communication patterns of individual jobs, routing, and placement of jobs. We propose two metrics that are indicators of the interference and performance variability that individual jobs can expect when executing on a system: *maximum % links shared* (MLS) and *total % links shared* (TLS). MLS is defined as the maximum over all jobs of the percentage of total number of links used by a job that are shared with other jobs (Eq. 5). This metric is an indicator of the interference encountered by the most impacted job in the workload. TLS is defined as the percentage of total number of links used by the workload that are shared among different jobs (Eq. 6). This metric is an indicator of the total interference caused by a workload. Consider a workload w that consists of jobs $j_0, j_1, \cdots, j_{m-1}$. Each job has nodes allocated to it using a predefined node allocation policy. Let l_i be the set of links used by messages associated with job j_i in the workload. We define:

$$MLS(w) = \max_{i \in [0,m)} \frac{\mathbf{card}\left(\left\{l_i \cap \left\{\cup_{k \in [0,m), i \neq k}\ l_k\right\}\right\}\right)}{\mathbf{card}(l_i)} \times 100 \tag{5}$$

$$TLS(w) = \frac{\mathbf{card}\left(\left\{\cup_{i,k \in [0,m), i \neq k}\ l_i \cap l_k\right\}\right)}{\mathbf{card}\left(\left\{\cup_{i \in [0,m)}\ l_i\right\}\right)} \times 100 \tag{6}$$

where **card** is the cardinality of a set. A network topology and associated node allocation policy that guarantee no inter-job interference should lead to an MLS and TLS of **zero**.

Figure 8 compares MLS and TLS for workloads that consist of all jobs running different instantiations of the Spread pattern at different node counts. Since communicating processes are selected randomly in Spread, given a sufficient number of workloads, we expect moving averages of MLS and TLS for the ensemble to give an approximation of the expected interference on a network. The node allocation policy used in these simulations is the clustered policy (Section III-B). We present results for G1 EM only since the partitionability characteristics of G1 and G2 EMs are similar.

Figures 8(a), (b) show that for all networks, some job in the workload shares a significant fraction of the links it uses with other jobs, and is highly likely to observe performance variability. The moving averages for both metrics stabilize with just 10 samples, which indicates that these values represent reasonable expectations for an arbitrary workload. Total interference statistics (Figures 8(c) ,(d)) follow similar trends as MLS.

Figures 9(a-d) show running averages for MLS and TLS when the workloads consist of different communication patterns. With adaptive routing, we observe that link sharing among jobs is high, especially for DF and SF. These results indicate that high performance variability is expected for all networks when clustered node allocation is used. This motivated us to explore alternative node allocation schemes that can help reduce inter-job interference.

B. Custom Node Allocation Policies

We now study each of the four networks to design custom node allocation policies that eliminate inter-job interference.

1) Dragonfly and Slim Fly: In DF and SF, non-minimal adaptive routing is necessary for good performance. This results in network links being shared among jobs if network traffic has to be sent outside a group (in DF) or sub-group (SF). Hence, as shown in Figure 11, the only policy that can eliminate inter-job interference requires limiting the maximum job size to node counts smaller than the size of a group or sub-group. In these cases, traffic generated by a job only uses links within its group and hence, does not share links with other jobs.

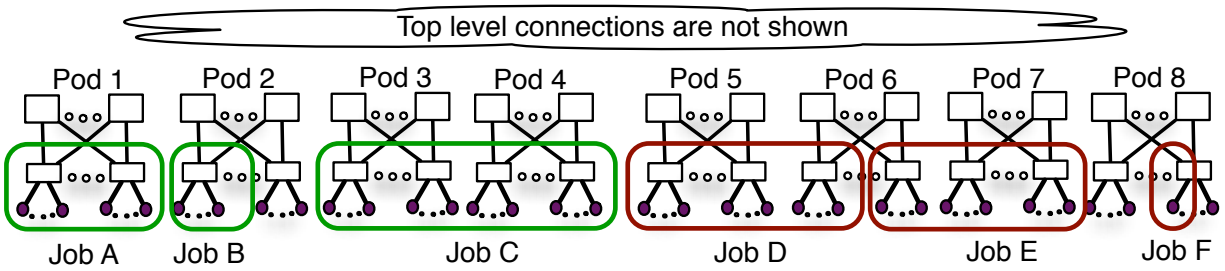


Figure 10: Partitioning a fat-tree: if each job is allocated either 1) all nodes connected to one or more switches in a single pod, or 2) all nodes connected to all switches in one or more pods, inter-job interference can be eliminated.

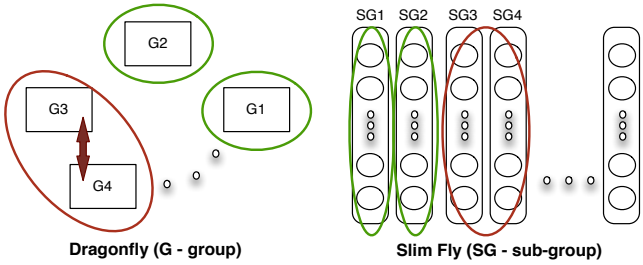


Figure 11: Partitioning dragonfly and Slim Fly: only if each individual job is contained within distinct groups and sub-groups respectively, inter-job interference can be avoided.

In a DF, jobs that are allocated multiple groups can also avoid performance variation but this requires use of static routing. Hence, allocating multiple groups to a job is not viable if we want to achieve high performance. Overall, for both DF and SF, the restriction of a job being limited to the size of a group significantly reduces the prospects of such a node allocation policy being used in production systems.

2) Fat-tree: In FT, leaf and intermediate switches are arranged in sub-groups called *pods*. The number of switches per pod is typically $r/2$, where r is the radix of a leaf switch. Leaf and intermediate switches that belong to a pod form a bipartite all-to-all graph. If a job is assigned all the nodes of a pod (Figure 10, Job A), it only uses links that connect leaf and intermediate switches within the pod. Hence, an allocation policy that always assigns an entire pod to an individual job avoids inter-job interference.

When a job, say X, is assigned nodes that span multiple pods (Jobs C, D, and E in Figure 10), in addition to using the links within those pods, traffic from job X may flow on links that connect the intermediate switches in these pods to *core* switches. If job X is assigned all the nodes on all switches of multiple pods (e.g. Job C), the only traffic that can flow through links connected to the intermediate switches of these pods is the traffic generated by job X. Moreover, on the core switches, the traffic generated by job X only contends for links and ports that are connected to the intermediate switches of the pods allocated to job X. Thus, if jobs are assigned all nodes of all switches of multiple pods, they do not interfere with other jobs. However, if a job is only assigned nodes of some switches in multiple pods (Jobs D and E), it may contend for resources with other jobs assigned to the same pod.

Within a pod, at the leaf level, if a job is assigned all the nodes connected to one or more switches, the ports and links connected to these switches are only used by the job and thus do not interfere with others (e.g. Job B in Figure 10). But, if a job is assigned only some nodes in one or more switches (Job F), it may interfere with other jobs. Finally, assignment of a job to nodes connected to a single leaf switch also does not cause any resource contention.

In summary, a node allocation policy with following restrictions can eliminate inter-job interference in FT:
i) job is assigned nodes connected to a single leaf switch, or
ii) job is assigned all nodes connected to one or more switches that belong to a single pod, or
iii) job is assigned nodes across multiple pods, but all other nodes in those pods are assigned to jobs that satisfy conditions listed in *i)* or *ii)*, or
iv) job is assigned all nodes connected to all switches in one or more pods.

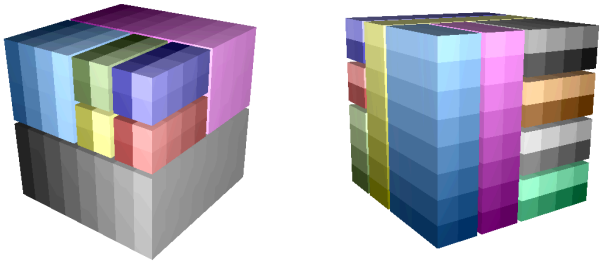
In comparison to DF and SF, interference-free node allocation policy for FT has fewer restrictions and allows for execution of jobs of various sizes. A practical implementation of this policy can be done by restricting large jobs to request node counts that are multiple of the number of nodes in a pod, and by restricting smaller jobs within pods.

3) Express mesh: Since EM is constructed by adding extra links to an n-D mesh along all dimensions, node allocation policies that have been used on production systems such as IBM Blue Gene [17], [12] and Cray XE/XK machines to create interference-free partitions can be directly applied to EM.

A node allocation policy that assigns convex shapes to each job can eliminate inter-job interference in EM, as in the case of meshes and tori. This is because when messages travel from a source to destination router both of which are within the convex shape allocated to a job, the traversal is always toward the destination in all dimensions. Within a dimension, the routing scheme described in Section II-C prohibits any changes in the direction of traversal. Hence, all the intermediate routers used for communication are also inside the convex shape, and dedicated to the job.

To obtain convex shapes for allocating jobs, the following two allocation policies have been successfully deployed on torus-based systems in the past and can be used for EM:

i) Building Blocks: In this scheme, a fixed-sized building block



(a) Building blocks (b) Variable-sized blocks

Figure 12: Partitioning express mesh: convex shape allocations eliminate inter-job interference (each color represents a job allocation; different shades used to show individual routers).

of routers is defined, e.g. of size $4 \times 4 \times 4$ or $8 \times 8 \times 1$. Every job allocation request is required to be a multiple of the total number of nodes in the fixed-size building block. Larger allocations are formed by combining building blocks located close to each other. Figure 12(a) shows an example of this type of allocation in which a $8 \times 8 \times 8$ EM is divided among jobs of various sizes using $2 \times 2 \times 2$ building blocks. Note that EM does not contain wrap-around links (unlike Blue Gene systems) and thus does not require extra links to enable partitioning.

ii) Variable-sized Blocks: This scheme relies on the resource manager dynamically computing sizes of cuboidal grids that can fit jobs of various sizes as they arrive, based on the current state of the system. Among the current torus-based systems, Blue Waters at NCSA deploys this scheme for reducing inter-job interference. Unlike the *building blocks* scheme, job allocation requests do not have size restrictions in this scheme. Figure 12(b) shows an example of this kind of allocation.

C. Impact of Partitioning

To analyze the impact of custom node allocation schemes described above, we create three sets of workloads. Each workload set further consists of six workloads. The six workloads within a set are chosen such that the first job is kept same in all of them, both in terms of its size and communication pattern. The sizes and patterns of the remaining jobs in the workloads are randomly chosen to fill up the rest of the system. Sub A2A, Stencil, and Spread are chosen as the constant first job in set 1, 2, and 3, respectively. We only consider medium- to large-sized jobs, with at least 5% of total nodes.

When custom node allocation policies are used for these workloads on EM and FT, we find that maximum % links shared (MLS) and total % links shared (TLS) are both zero. This suggests that custom node allocation policies should alleviate performance variability on EM and FT.

Figure 13 (top) presents the predicted communication rate for the jobs kept constant within each workload set. Labels FT* and EM* represent results for custom node allocation, while the remaining labels use clustered node allocation. 3D EM with $g = 1$ is used as a representative of EM topology. The figure shows that when custom node allocation is used, performance variability caused by inter-job interference is eliminated and

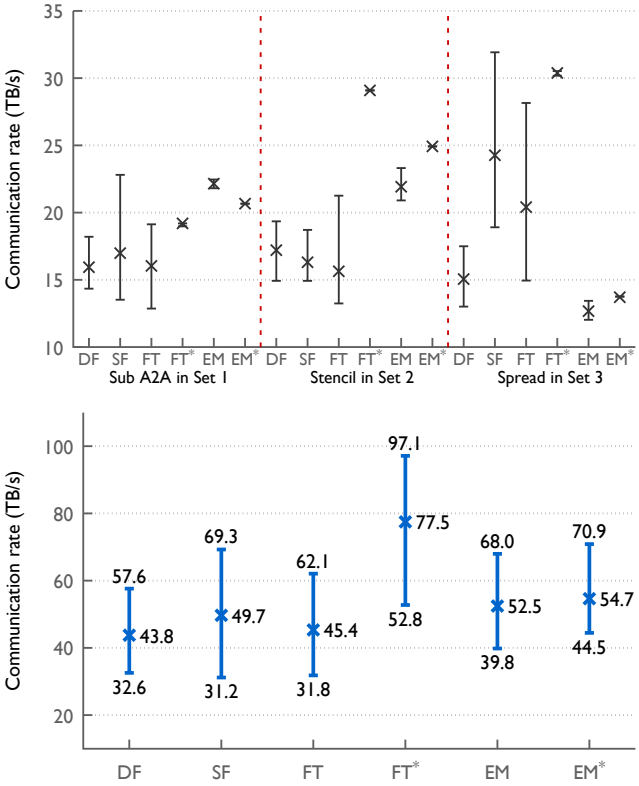


Figure 13: Impact of partitioning: FT* and EM* represent simulations in which interference-free node allocation schemes are used. Minimum, average, and maximum values for the job kept constant across different workloads within a set are shown (top). Values shown are aggregated communication rates across 30 different workloads used in Figure 7 (bottom).

consistent communication rate is predicted for the job kept constant on EM and FT.

Figure 13 (bottom) shows that use of custom node allocation does not impact the predicted communication rate negatively. In fact, for FT*, use of custom node allocation improves performance significantly. Overall, when we compare the median communication rate for all workloads, we find a 71% improvement in the communication rate on FT* because of the use of interference-free partitioning. On EM* also, the median communicate rate increases by 7%. These results suggest that customized node allocation not only eliminates performance variability, it also results in performance improvement.

V. Related Work

Several topologies have been proposed and implemented for HPC networks – hypercube [11], express cube [**?**], fat-tree [1], torus [17], [12], dragonfly [2], [13], HyperX [9], Slim Fly [4], and others. The key aim in many of these network designs is to minimize network diameter by exploiting high-radix routers. Our focus is on constructing networks that are partitionable. The construction of EM is most similar to express cube but aims at creating low-diameter mesh-based networks. For gap $= 1$, EM reduces to HyperX topology, whose use is limited

by the maximum system size it can support if router radix is fixed. In addition to the publications mentioned above, several studies have used simulations to study and compare different network topologies [15], [18], [19], [20]. Our work uses tools and methodologies proposed in some of these publications.

Effect of inter-job interference has been analyzed in several recent publications [6], [7], [8]. The ability of torus-based machines such as Blue Gene systems [17], [12] to provide interference-free execution is well known. This work builds upon the concepts used in these machines to eliminate inter-job interference on EM. We also use ideas from Puente et al. [10] to design deadlock-free routing schemes for EM.

VI. Conclusion

Network-oblivious resource allocation and non-minimal routing on LDN topologies used in HPC systems increase inter-job interference that causes performance variability and negatively impacts the overall performance of the system. Thus, techniques that can eliminate inter-job interference and improve overall performance will be important for efficient utilization of next-generation HPC systems.

In this paper, we showed that express mesh and fat-tree are two LDN topologies that can address the challenge of inter-job interference. Express mesh can guarantee performance predictability by utilizing node allocation policies deployed in systems such as Blue Gene and XE/XK. For fat-tree, we presented a new node allocation policy to provide interference-free partitions. We also showed that these topologies match the performance of other LDN topologies such as dragonfly and Slim Fly. Further, the procurement cost of express mesh is expected to be lower than fat-tree, similar to dragonfly, and higher than Slim Fly.

This work also showed that node allocation policies that eliminate inter-job interference can result in higher overall performance. We observed a 7% performance increase for multi-job workloads on express mesh by using a Blue Gene style node allocation policy. On fat-tree, we showed that interference-free node allocation policy can result in performance gains of up to 71%. We hope that these findings will help network architects and developers of resource managers in improving throughput of current and future HPC systems.

Acknowledgment

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-706801).

References

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

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

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

[4] M. Besta and T. Hoefler, "Slim Fly: A cost effective low-diameter network topology," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. IEEE Press, 2014, pp. 348–359.

[5] M. A. Jette, A. B. Yoo, and M. Grondona, "SLURM: Simple linux utility for resource management," in *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*. Springer-Verlag, 2002, pp. 44–60.

[6] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs, "There goes the neighborhood: performance degradation due to nearby jobs," in *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. IEEE Computer Society, Nov. 2013, LLNL-CONF-635776.

[7] S. A. Smith, D. K. Lowenthal, A. Bhatele, J. J. Thiagarajan, P.-T. Bremer, and Y. Livnat, "Analyzing inter-job interference in dragonfly networks," 2017, under review.

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

[9] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "HyperX: Topology, routing, and packaging of efficient large-scale networks," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09. New York, NY, USA: ACM, 2009.

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

[11] J. P. Hayes, T. N. Mudge, and Q. F. Stout, "Architecture of a hypercube supercomputer," in *ICPP*, 1986, pp. 653–660.

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

[13] G. Faanes, A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard, "Cray Cascade: A scalable HPC system based on a dragonfly network," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012.

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

[15] N. Jain, A. Bhatele, S. T. White, T. Gamblin, and L. V. Kale, "Evaluating HPC networks via simulation of parallel workloads," in *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '16. IEEE Computer Society, Nov. 2016, LLNL-CONF-690662.

[16] C. Huang, G. Zheng, S. Kumar, and L. V. Kalé, "Performance Evaluation of Adaptive MPI," in *Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming 2006*, March 2006.

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

[18] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, Jun. 2014.

[19] K. Underwood, M. Levenhagen, and A. Rodrigues, "Simulating Red Storm: Challenges and successes in building a system simulation," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS '07)*, 2007.

[20] A. Bhatele, N. Jain, W. D. Gropp, and L. V. Kale, "Avoiding hot-spots on two-level direct networks," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. ACM, Nov. 2011, LLNL-CONF-491454.