Maximizing Throughput on a Dragonfly Network

Nikhil Jain*, Abhinav Bhatele†, Xiang Ni*, Nicholas J. Wright‡, Laxmikant V. Kale*

*Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801 USA
†Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, California 94551 USA
‡NERSC, Lawrence Berkeley National Laboratory, Berkeley, California 94720 USA
E-mail: nikhil@illinois.edu, bhatele@llnl.gov, xiangni2@illinois.edu, njwright@lbl.gov, kale@illinois.edu

***Abstract*—Interconnection networks are a critical resource for large supercomputers. The dragonfly topology, which provides a low network diameter and large bisection bandwidth, is being explored as a promising option for building multi-Petaflop/s and Exaflop/s systems. Unlike the extensively studied torus networks, the best choices of message routing and job placement strategies for the dragonfly topology are not well understood. This paper aims at analyzing the behavior of an interconnect based on the dragonfly topology for various routing strategies, job placement policies, and application communication patterns. Our study is based on a novel model that predicts traffic on individual links for direct, indirect, and adaptive routing strategies. We analyze results for individual communication patterns and some common parallel job workloads. The predictions presented in this paper are for a 100+ Petaflop/s prototype machine with 92,160 high-radix routers and 8.8 million cores.**

I. Introduction

HPC systems are typically associated with low latency and high bandwidth networks which lead to fast messaging between communicating processes. However, as systems become larger and computation continues to become cheaper due to many-core nodes with accelerators, the network is increasingly becoming a scarce resource. This has pushed the HPC community towards designing low-diameter fast networks with large bisection bandwidth. The dragonfly topology [1] and its variants [2], [3], [4] are actively being explored as the interconnects that satisfy all these requirements.

In the dragonfly topology, high-radix routers are used to organize the network into a two-level all-to-all or closely connected system. The presence of these multi-level hierarchies connected through network links opens up the possibilities for different routing strategies and job placement policies. However, unlike the extensively studied torus network, the best choices of message routing and job placement policies are not well understood for the dragonfly topology.

This paper compares various techniques to maximize the network throughput on a 100+ Petaflop/s prototype machine with a dragonfly interconnect comprised of 92,160 routers and 8.8 million cores. We evaluate the proposed system using a congestion aware model for network link utilization. A wide variety of routing strategies for the dragonfly topology are compared – static direct, static indirect, adaptive direct, adaptive indirect, and adaptive hybrid routing. To the best of our knowledge, this is the first work that predicts network utilization for adaptive routings on the dragonfly interconnect. The predictions are performed for various combinations of routing strategies with different job placement policies – random or round-robin allocation of nodes, routers, chassis or groups to each job.

The unusually long time spent in PDES-based network simulations in our previous work [5] prompted us to use analytical modeling in this paper. We have developed a message-level, congestion-aware iterative model of the dragonfly topology that predicts network link throughput. A parallel MPI implementation of the model has been used to perform the experiments. The model is used to determine the link throughput for various routing strategy and job placement combinations and answer questions such as:

- What is the best combination for single jobs with communication patterns such as unstructured mesh, 4D stencil, many-to-many, and random neighbors? These patterns represent production scientific applications routinely run on NERSC machines [6], [7].
- What is the best combination for parallel job workloads in which several applications are using the network simultaneously?
- Is it beneficial for jobs in a workload to use different routing strategies that are more suitable for them in isolation? What is the best placement policy in this situation?

To the best of our knowledge, such studies have not been reported so far for a dragonfly network. We believe that the analysis presented in this paper will be useful to application end-users in identifying good configurations for executing their applications on a dragonfly interconnect. At the same time, these results can be used by machine architects and system administrators to decide the best default job placement policies and routing strategies.

II. The Dragonfly Interconnect

Multi-level direct networks have been proposed recently by several researchers as a scalable topology for connecting a large number of nodes together [1], [2], [3], [4]. The basic idea behind these networks is to have a topology that resembles an all-to-all at each level of the hierarchy which gives the impression of a highly connected network. Further analysis would show that the network is built using high-radix routers that only exist at the lowest level. The connections between these routers create an appearance of several all-to-all connected direct networks at multiple levels of the hierarchy.

Two prominent implementations of multi-level direct networks are the PERCS interconnect by IBM [3] and the Cascade

SC14, November 16-21, 2014, New Orleans
978-1-4799-5500-8/14/$31.00

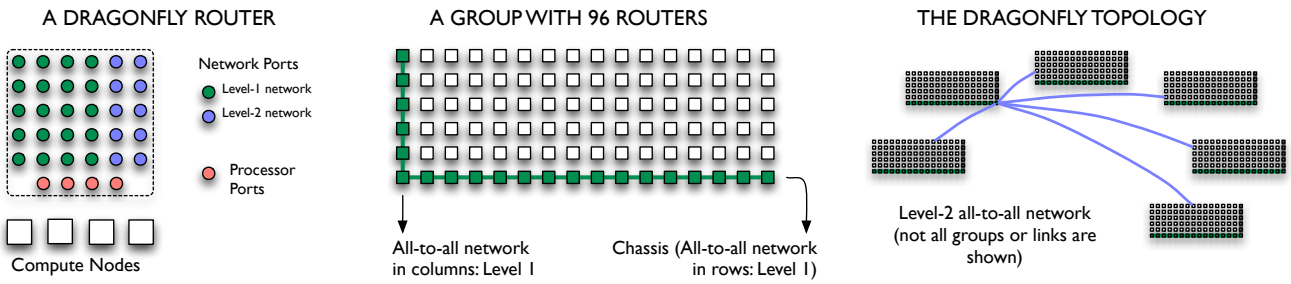

Fig. 1: The structure of a dragonfly network

system by Cray [4]. We focus on the Cascade system which is based on the dragonfly topology designed by Kim et al. [1]. The Cascade (Cray XC30) system uses the Aries router as its building block and has been used in supercomputers such as Edison at NERSC, Lawrence Berkeley National Laboratory and Piz Daint at the Swiss National Supercomputing Centre.

In this paper, we use the dragonfly topology to build a prospective 100+ Petaflop/s system. The parameters for this prototype machine are inspired by the Cray Cascade system. We have, however, simplified the router and link bandwidths for ease of modeling. The building block is a network router with 30 network ports and 4 processor ports (Figure 1). Each network router is connected to four compute nodes (of 24 cores each) through the processor ports. Sixteen such routers form a chassis and six chassis are combined together to form a group ($16 \times 6 = 96$ routers in total). Each network router is connected to all other routers in its chassis (15 ports) and to the corresponding routers in five other chassis (5 ports). These links along rows and columns in the group are called level 1 (L1) links in this paper. The remaining 10 ports are used to connect to network routers in other groups. These inter-group links form the second level (L2) of the network. L1 and L2 links together form a two-level direct network.

We take 960 such groups comprised of 96 routers (384 nodes) each to build a very large dragonfly system. This machine has 8,847,360 cores (8.8 million) and extrapolating the Edison system — a peak performance of 164.5 Petaflop/s. Two major differences between the prototype machine used in the paper and the Cray Cascade system are: 1. There is only one L1 link between each pair of routers along the column whereas the Cascade machine has three such links leading to three times the bandwidth in that dimension, 2. Cray only allows for 240 groups which leads to 4 links connecting each pair of groups and hence much higher bandwidth.

Related Work: Formal models such as LogP [8] and LogGP [9] have been used to analyze the communication in parallel applications for a long time. Subsequently, based on the LogP model, models such as LoPC [10], LoGPC [11], LoGPG [12], LogGPO [13], and LoOgGP [14] were developed to account for network congestion. Unlike the model in this paper, these models do not consider routing protocols to model congestion and do not model the traffic on individual links. Simulators based on these models, e.g. LogGOPSim [15], simulate application traces and are closer to our work.

Hoefler et al. [16] developed models for the traffic on individual links in the presence of congestion for three different network topologies – 3D torus, PERCS and Infiniband. Bhatele et al. used BigSim [17], a discrete-event simulator to study application performance under different task mappings and routings on an IBM PERCS machine [5]. The unusually long time spent in each BigSim simulation prompted the authors to use analytical modeling in this paper. Chakaravarthy et al. [18] present a formal analysis of the mappings proposed in our previous publication [5] and some new mappings.

Three things distinguish this work from the previous communication and congestion modeling work. First, we consider different alternative routings with adaptivity and study their impact on network throughput. Second, we consider representative job workloads at supercomputing sites and simulate different routings and job placement strategies for these workloads. Third, this paper presents analysis for the dragonfly network at an unprecedented scale (8.8 million cores).

III. Prediction Methodology for Link Utilization

Modeling is a powerful tool to explore design choices for future systems; it is also useful for analyzing scenarios that are challenging or expensive to deploy on existing systems. We present a model and its implementation to predict network throughput for dragonfly networks.

A. Prediction Model

In order to compare the relative benefits of different job placement policies and routing strategies, we have developed a model that generates the traffic distribution for all network links given a parallel communication trace. Our hypothesis is that the traffic distribution is indicative of the network throughput we can expect for a given scenario [5], [19], [20]. The inputs to this model are:
— A network graph among dragonfly routers, $N = (V, E)$.
— An application communication graph for one time step or phase in terms of MPI ranks, $A^C = (V^C, E^C)$.
— A job placement/mapping of MPI ranks to physical cores.
— A routing strategy, $\Re$.

The model accounts for contention on network links and outputs the expected traffic on all network links for each phase of the application. All communication in one time step or phase is assumed to be occurring simultaneously on the network and all messages for the phase are considered to be in flight. For each phase, an *iterative solve* is performed to get the probabilistic traffic distribution on the links. Only one iteration may be needed for simple cases, such as the direct routing. The iterative solve in the model is described below.

Initialization: The input network graph N gives us the peak bandwidths on all network links. We define two other copies

of this graph – $N^A = (V^A, E^A)$, which stores the bandwidths that have already been allocated to different messages; and $N^R = (V^R, E^R)$, which stores the remaining link bandwidths that can still be allocated in subsequent iterations. For edge l in these graphs, this relationship holds: $E_l = E_l^A + E_l^R$. At initialization, $E_l^A = 0$ and $E_l^R = E_l$ for all edges.

Iteration: The **do** loop below describes the iterative solve which is central to our traffic prediction model:

do *until no message is allocated any additional bandwidth*

1) For each edge (message), m in E^C, obtain a list of paths, $P(m)$ that it can send its packets on from the source to the destination router for a given routing $\Re$.
2) Derive the "request" count for each link using the $P(m)$ sets for all messages. The request count is the total number of messages that want to use a link; store the request counts for all links in another copy of the network graph, $N^{RC} = (V^{RC}, E^{RC})$.
3) For each path, p in $P(m)$ for each message m in E^C, calculate the "availability" of each link in p. Availability of a link l is its remaining bandwidth divided by its request count, E_l^R / E_l^{RC}. Each link on path p allocates additional bandwidth to message m which equals the minimum of the availabilities of all links on that path.
4) Decrement remaining bandwidth values in N^R and increment values in N^A based on the freshly allocated bandwidths on the links in the previous step.

end do

Post Processing: For each message, the model assumes that its packets will be divided among the paths on which it was allocated bandwidth during the iterative solve. Depending on the routing protocol $\Re$, the fraction of a message that is sent on different paths is computed differently. Thus, we obtain the traffic on a link l as,

$$\textit{traffic}(l) = \sum_{\forall m \in E^C} f_p \ \textit{if} \ l \ \in p, \forall \ p \in P(m)$$

where f_p is the fraction of the message assigned to path *p* in the set $P(m)$.

This iterative model is generic and can be used for any routing by selecting appropriate schemes for finding $P(m)$ in Step 1, deciding the request counts N^{RC} in Step 2, finding the link availability in Step 3, and deciding the f_p in post processing. The specific schemes used for different routings are described in detail in the next section.

B. Parallel Network Routing Prediction

The model described in the previous section has been implemented as a scalable MPI-based parallel program. For most parts, the parallelism is obtained by dividing the processing of the messages among the MPI processes. The implementations for different routing schemes build upon the generic model and customize it to improve the prediction capability and computation time. In the following description of the routing schemes that are based on schemes proposed by Kim et al. [1], it is assumed that a message is sent from the source router *s* to the destination router *d*.

Static Direct (SD): In this scheme, a message from *s* to *d* is sent using the shortest path(s) between *s* and *d*. If multiple shortest paths are present, the message is evenly divided among the paths. For the dragonfly interconnect described in Section II, the maximum number of hops for SD routing is 5 — two L1 hops in the source group, one L2 hop, and two L1 hops in the destination group.

For the evaluation of SD, only one iteration is needed to find all shortest paths that a message can take. Once those paths are determined, the message is divided equally among those paths during the post processing. Note that since this routing does not make use of the request count and availability computed in Step 2 and Step 3 respectively, our implementation skips those steps of the iteration.

Static Indirect (SI): In this scheme, for each packet created from a message, a random intermediate router *i* is selected. The packet is first sent to *i* using a shortest path between *s* and *i*. It is thereafter forwarded to *d* using a shortest path between *i* and *d*. For the given interconnect, use of an intermediate router results in the maximum number of hops for SI to be 10.

Ideally, for packet-level SI routing, only one iteration is needed to find all the indirect paths (like direct routing). However, storing all indirect routes requires very large amount of memory. To address the memory concern, our implementation goes over the packets in the message one by one, and assigns them to a randomly generated indirect path. Processing each packet individually leads to extremely high workload making this routing the most time consuming to evaluate.

Adaptive Direct (AD): The AD routing adds adaptivity to SD — if multiple shortest paths are present between *s* and *d*, the message is divided among the paths based on the contention on those paths. The iterative solve is suitable for adaptive routing given that it allows a message to request more bandwidth on resources that have leftover bandwidth iteratively. It also allows messages that can use multiple paths to get more bandwidth. In a typical run, we ran the iterative solve till convergence is reached, i.e. no message is able to obtain any more bandwidth for any of its paths.

Customization: In Step 2, instead of assigning equal weights to all requests of a message to the links of the paths it can use, the requests are weighted based on the minimum remaining bandwidth on any link of the paths. For example, if a message could be sent on two paths with 50 and 100 units of minimum remaining bandwidth on the links of those paths respectively, the requests to the links on those paths are given weights 0.33 and 0.66 respectively. Such weighted requests are helpful in adaptively selecting links that are less congested. Also, the size of a message is considered while deciding the weights of the requests. This allows for favoring larger messages which may increase the overall throughput of the network as described next. In Step 3, on receiving several requests for a link from various messages, instead of equally dividing the remaining bandwidth to all requests, the division is weighted based on the weights of the requests. During post processing, the messages are divided among the paths in proportion to the bandwidth allocated on those paths so that the effective traffic on all links is equalized (as opposed to the static division done by SD).

Adaptive Indirect (AI): The AI routing is related to SI routing in a manner similar to the relation between SD and AD. For each packet sent using AI routing, the intermediate router, *i* ,

is selected from a randomly generated set of routers, based on the contention on the corresponding paths.

Customization: The implementation for this routing also uses the schemes described for adaptive direct routing. However, while adaptive direct routing uses the same set of paths in every iteration for a message, it is impractical to use thousands of paths in every iteration as required by the indirect routing. As a result, we used a set of 4 indirect paths selected afresh in every iteration. However, this may overload the links of the paths used in initial iterations since more bandwidth is typically available during the start. In order to overcome this bias, we added the concept of *incremental* bandwidth. In this method, at the very beginning, only a fraction of the maximum bandwidth of the links is available for allocation to the messages. In each iteration, more bandwidth is made available incrementally for allocation. This kind of increment of available bandwidth is continued until we have exposed all of the maximum bandwidth of the links. In our experiments, we exposed an additional fraction ($\frac{1}{f}$) of bandwidth in each of the first f iterations. Prediction results with varying f suggested that beyond $f = 50$, incremental exposure of bandwidth has no effect on the predictions.

Adaptive Hybrid (AH): A hybrid of AI and AD leads to the AH routing. In this scheme, for sending each packet, the least contended path is selected from a fixed size set of shortest paths and indirect paths. The indirect paths in the set are generated randomly for every packet of the message. AH is implemented using the same schemes as described for AI. To allow for use of direct paths in each iteration, the set of paths consists of 4 paths — up to two direct paths and the remaining indirect paths, instead of 4 indirect paths used for AI. This helps in biasing the routing towards direct paths if congestion on them is not high. In the current implementation of the model, we have assumed global knowledge of congestion (e.g. a router can estimate queue lengths on other routers). Hence, in terms of the original terminology used by Kim et al. [1], the model predicts link utilization for UGAL-G routing, which is an ideal implementation of Universal Globally-Adaptive Load-balanced (UGAL) routing.

IV. Evaluation Setup

A suitable routing and an intelligent job placement can be used to efficiently utilize the links of a dragonfly interconnect. For the routings described in Section III-B, we study the dragonfly interconnect using the presented prediction framework for many job placement policies and communication patterns. In this section, we briefly describe these job placement policies, list the communication patterns, and explain the experimental set up.

A. Job Placement

Job placement refers to the scheduling scheme used to assign a particular set of cores in a particular order for execution of a job. The ordering of the cores is important because it determines the mapping of MPI ranks to the physical cores. We explore the following schemes that have been chosen based on our previous work on two-tier direct networks [5] and the schemes that are currently in use at supercomputer centers that host Cray XC30, a dragonfly interconnect based machine.

TABLE I: Details of communication patterns.

Communication Pattern	Number of Processes	Messages per Process (TDC)	Message Size (KB)
Unstructured Mesh	8,847,360	6 - 20	512
Structured Grid	$80 \times 48 \times 48 \times 48$	8	2,048
Many to many	$180 \times 128 \times 384$	127	100
Uniform Spread	8,847,360	6 - 20	512

Random Nodes (RDN): In this scheme, the job is allocated randomly selected nodes from the set of all available nodes in the system. The cores of a nodes are ordered consecutively, while the nodes are ordered randomly. Random placement may be helpful in spreading the communication uniformly in the system, thus resulting in higher utilization of the links.

Random Routers (RDR): The RDR scheme increases the level of blocking by allocating randomly selected routers (set of four nodes) to a job. The cores attached to a router are ordered consecutively, but the routers are ordered randomly. The additional blocking may help in restricting the communication leaving the router. It also avoids contention within a router among different jobs running on different nodes of the router.

Random Chassis (RDC): This scheme allocates randomly selected *chassis* to a job. The cores within a chassis are ordered, but the chassis are randomly arranged. The additional blocking may limit the number of hops to one L1 link for the messages of a job with communicating nearby MPI ranks.

Random Groups (RDG): The RDG scheme further increases the blocking to groups. This may be useful in reducing the average pressure on L2 links by restricting a significant fraction of communication to be intra-group. However, it may also overload a few L2 links if the groups connected by a L2 link contains nearby MPI ranks that communicate heavily.

Round Robin Nodes (RRN): In this scheme, a job is allocated nodes in a round robin manner across the groups. The cores of a nodes are ordered consecutively, while the nodes are ordered in a round robin manner. Such a distribution ensures uniform spreading of a job in the system.

Round Robin Routers (RRR): The RRR scheme is similar to the RRN scheme, but allocates routers instead of individual nodes to a job in a round robin manner.

B. Communication Patterns

Kamil et al. [21] have defined topological degree of communication (TDC) of a processor as the number of its communication partners. They study a large set of important applications and show that the TDC of common applications vary from as low as 4 to as large as 255. In order to span a similar range of TDC and study a representative set of common communication patterns [6], [7], the patterns listed in Table I have been used. Each of the pattern is described in more detail as we analyze prediction results for it in Section V.

The communication graphs for each of the pattern is generated either by executing them using AMPI [22], which allows us to execute more MPI processes than the physical cores, or by using a simple sequential program that replicates the communication structure of these patterns.

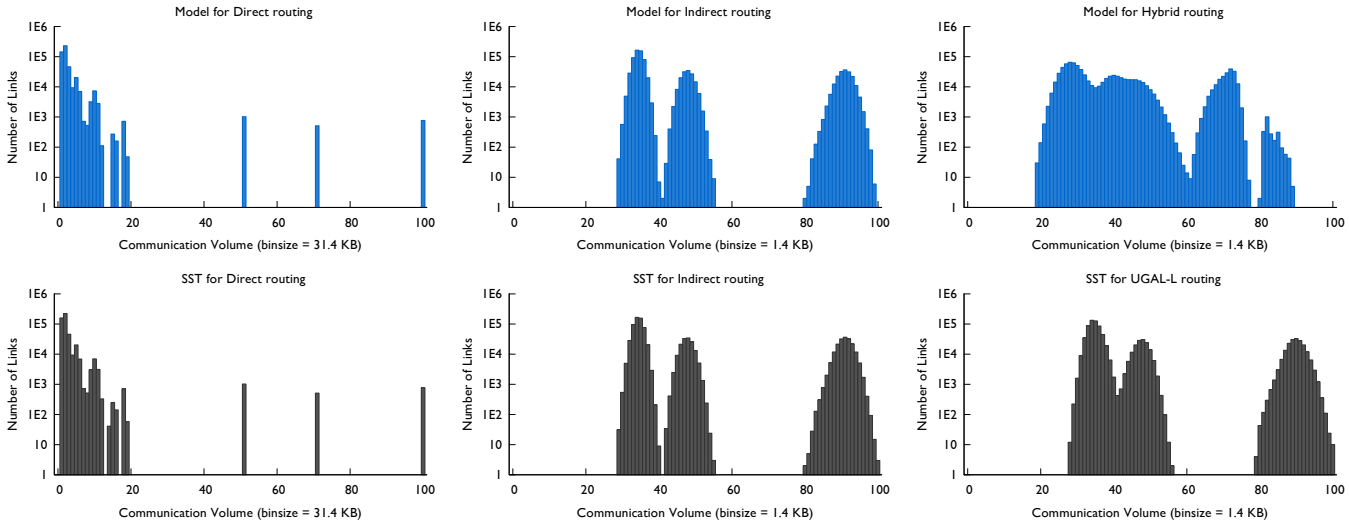

Fig. 2: Comparison of the predictions by the presented model with predictions by SST/macro, a packet-level simulator, for a 4D Stencil simulated on a $36,864$ router system.

C. Prediction Runs

The parallel code that implements the presented model was executed on Vesta and Mira, IBM Blue Gene/Qs at ANL and Blacklight, an SGI UV shared-memory system at PSC. For each run, three input parameters were provided: 1) communication pattern based on MPI ranks, 2) mapping of MPI ranks to physical cores, 3) system configuration including the routing strategy. Depending on the communication pattern and the routing, different core counts were used for runs. Typically, for SD and AD routing schemes, 512 cores were used to complete the simulation in $\approx$ 5 minutes. For the remaining routings, $2,048$ cores were used to simulate the lighter communication patterns, such as structured grid, in up to ≈ 30 minutes. For heavy communication patterns, e.g. many to many, $4096 - 8192$ cores were required to finish the runs in up to two hours.

D. Model Comparison

In order to verify the accuracy of the presented model, we compare the predictions made by the model with the predictions by SST/macro, a packet-level discrete-event simulator [23]. For these comparisons, we use near-neighbor communication pattern of an application that performs four-dimensional stencil computation. The prototype system considered here is relatively small ($36,864$ routers with one active MPI rank per router), so that the predictions using SST/macro can be obtained in a reasonable time frame.

The left graph in Figure 2 shows the histograms of the predicted traffic distributions for direct routing using our model and SST/macro. The two histograms are very similar which attests that the presented model closely resembles the predictions of a packet-level simulation for direct routing. Similar results are seen for indirect routing (center graph in Figure 2) which validates the model for indirect routing. For hybrid routing, we were not able to use SST/macro for a direct verification because it implements UGAL-L (a localized version of UGAL), while our model assumes global knowledge. Nevertheless, we present the predictions by SST's UGAL-L and our model's routing schemes in the right graph in Figure 2. We observe that the predictions by SST's UGAL routing are very similar to its predictions using indirect routing. This is possibly due to the localized view of the queues on a router; the queues for direct routes gets filled up quickly for large messages, hence diverting the traffic towards indirect routes. In contrast, the hybrid model is able to offload heavily used links (due to its global knowledge) and shift many links to left bins in comparison to the indirect routing.

V. Predictions for Single Jobs

The first half of the experiments are focused on understanding network throughput for single job execution on the dragonfly interconnect. We begin this section with a brief guide on how to analyze the box plots presented in the rest of the paper. Following it, the four communication patterns are studied in detail. Finally, we present prediction results for the case in which the many-to-many pattern is executed in isolation on the system with variation in the number of cores used by it.

A. Description of the Plots

Figure 3 shows a typical box plot used in this paper. The x-axis contains combinations of routing strategies and job placement policies, which are grouped based on the routing strategy. The log scale based y-axis is the amount of traffic flowing on links in megabytes. For each combination of job placement and routing, six data points are shown — the minimum traffic on any link, the first quartile – 25% of links have lesser traffic than it, the median traffic, the average traffic on all the links, the third quartile – 75% of links have lesser traffic than it, and the maximum traffic on any link. The plot also shows a horizontal dotted blue line that indicates the lowest maximum traffic among all the combinations.

Very high value of maximum traffic relative to other data point indicates network hotspots . Hence, it is a good measure to identify scenarios whose throughput is impacted by bottleneck link(s). The average traffic is an indicator of

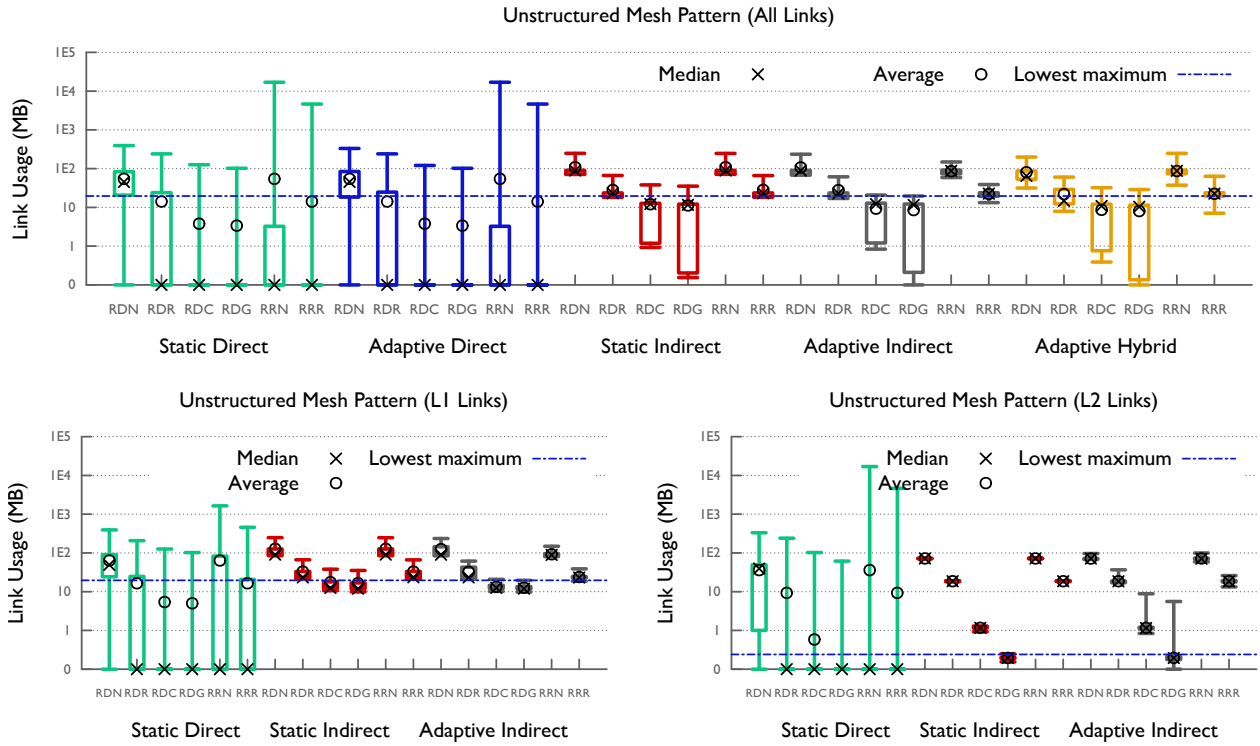

Fig. 4: Unstructured Mesh Pattern (UMesh): blocking helps in improving the traffic distribution.

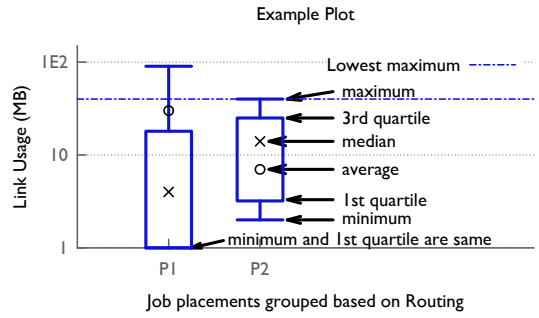

Fig. 3: Example to explain the data displayed in the plots.

the overall load on the interconnect. It is helpful in finding scenarios that reduce total traffic and hops taken by the messages. Comparing the average with median is valuable for estimating the distribution. If average is significantly higher than the median (P1 in Figure 3), the distribution is skewed to the right — most of the links have relatively low traffic, but a long tail stretches to the right. In contrast, if median is higher than the average, the distribution is skewed to the left — most of the links have more traffic than the average, but a long tail stretches to the left. Finally, the quartiles can be used to find more information about how much fraction of the links had what volume of traffic flowing through them. Overall, we suggest that *a distribution with closer values of these data points is good for network throughput*. In case of similar distributions, lower values are better for throughput.

B. Unstructured Mesh Pattern (UMesh)

In this pattern, each MPI process r communicates with $6-20$ other MPI processes in its neighborhood (within range [r-30, r+30]). Such a pattern is representative of unstructured mesh based and particle in cell (PIC) codes with space filling curve based mapping of MPI processes (e.g. Morton ordering).

Effect of Job Placement: Figure 4 (top) presents the expected link utilization when UMesh is executed on the full system. It can be seen that as we increase the blocking in job placement, the maximum, the average, and the quartiles decrease significantly. For UMesh with many communicating nearby MPI ranks, this trend is observed because increasing blocking from nodes to router avoids network communication. Additionally, it may also decrease the number of hops traversed by messages, since it places most communicating MPI processes within a chassis or a group (as we move from RDR to RDC and RDG).

Effect of Indirect Routing: Comparison among routings shows that the use of any form of indirect routing leads to an increase in average traffic on the links, a trend that is seen in all results presented in this paper. This is expected since indirect routing forces use of extra hops. However, indirect routing also leads to a more uniform distribution of loads on the links which is demonstrated by the closes values of the quartiles. Also, the median is closer to the average for indirect routing, in contrast with direct routing for which median is mostly zero (indicating a distribution skewed to the right). Note that although indirect routing increases the average, owing to a better distribution, the maximum is never worse than the direct routings for a given job placement. These characteristics indicate better network throughput for indirect routing in comparison to direct routing.

We also observe that for direct routing with RRN and RRR

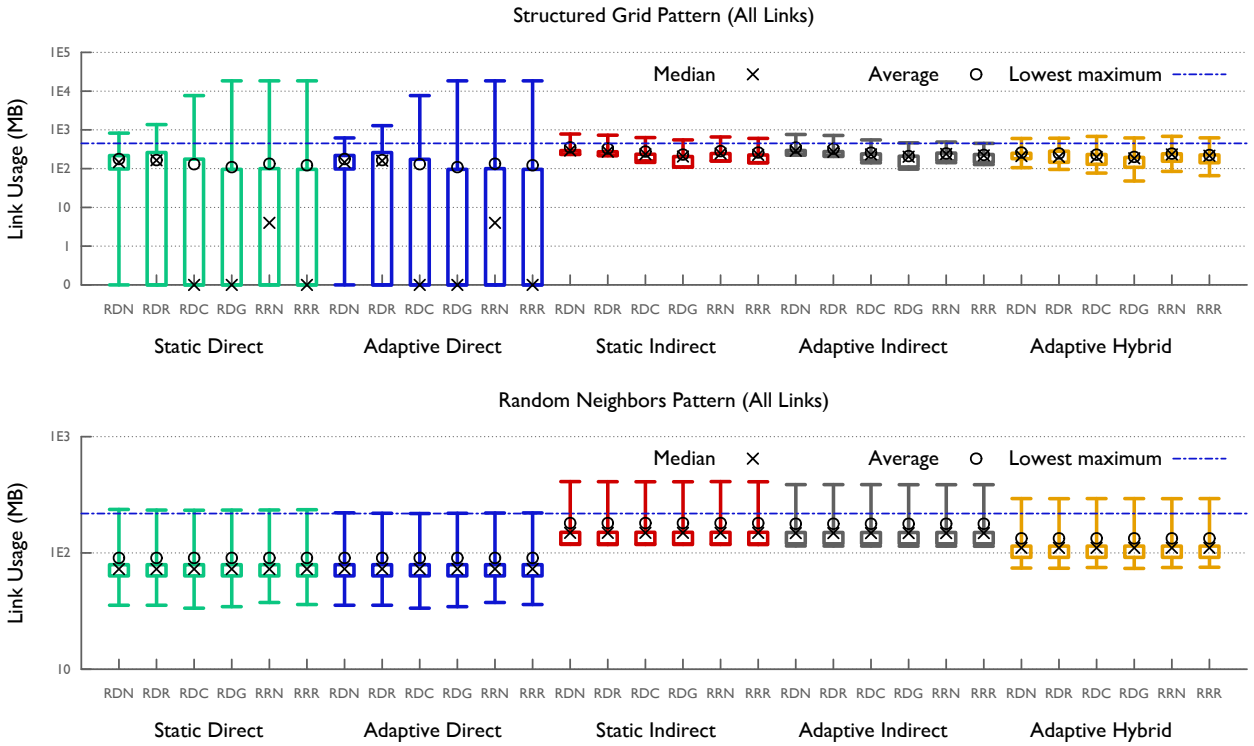

Fig. 5: Structured Grid Pattern (4D Stencil) and Random Neighbors Pattern (Spread)

placements (shown for SD in Figure 4 (bottom)), only a few L2 links are being used heavily, thus increasing the overall maximum. These are the L2 links that connect the consecutive groups which are used by the communication among nearby MPI ranks mapped to the nodes and routers placed in a round-robin manner. Indirect routing offloads these L2 links by distributing the traffic to other unused L2 links.

Effect of Adaptivity: We observe that the expected traffic for adaptive versions of the routing schemes have very similar distribution to the static version with similar or lesser corresponding values for the data points of interest. In particular, for RDC and RDG, the AI routing scheme reduces the maximum traffic by 50% in comparison to its static counterpart, SI. We attribute this improvement to unloading of overloaded L1 links. As shown in Figure 4 (bottom), comparison of the average suggests that the L1 links are more loaded which is expected given the dominant nearby MPI rank communication in UMesh. For RDC and RDG, the AI routing is able to improve the distribution of traffic on L1 links, and thus reduces the maximum traffic.

C. Structured Grid Pattern (4D Stencil)

Based on a four-dimensional nine-point stencil, this pattern is representative of the communication pattern in MILC, a Lattice QCD code [24]. The MPI processes are arranged in a 4-D grid, with each process communicating with its 8 nearest neighbors in the four dimensions. As a result, this pattern has lesser MPI rank based communication locality in comparison to UMesh. For 4D Stencil, two of an MPI process' communicating partners are its immediate MPI rank neighbors, but the remaining six neighbors are placed incrementally further away from it. For the configuration provided in Table I, two of the neighbors are 48 MPI ranks away, the next pair is $2,304$ ranks away, and the final two are $110,592$ ranks away.

Effect of Job Placement: Figure 5 (top) shows the traffic distribution predictions for 4D Stencil. For direct routings, in a manner similar to UMesh, the average and the quartiles decrease as blocking is increased, although the decrease in average is significantly lesser when compared to UMesh. However, in contrast to UMesh, the maximum traffic increases as we increase the blocking. We suspect that the increase in the maximum is due to high traffic on a few L2 links — links that connect groups which contain many pairs of communicating MPI processes. Such scenarios may arise when blocking is performed at chassis and group levels. In this case, communication between corresponding consecutive MPI processes in two sets that are roughly 48, 2304, or $110,592$ MPI ranks apart may result in large number of communicating pairs, thus overloading a few L2 links. To verify this assumption, we first studied the histogram for L2 link utilization (shown in Figure 6). It can be seen that while most of the L2 links are unused, a few are overloaded. Then, we identified these links using the raw link usage data and found them to be suspected links, hence verifying our assumption.

Effect of Indirect Routing: The skewness caused by the overloading of a few L2 links for direct routing is eliminated by the use of indirect routing. As shown in Figure 5 (top), indirect routing leads to a better distribution of traffic on the links. However, as we saw for UMesh, it also increases the average traffic on the links. These results are consistent with our past work on two-level direct networks in which 4D Stencil was also used as communication pattern [5].

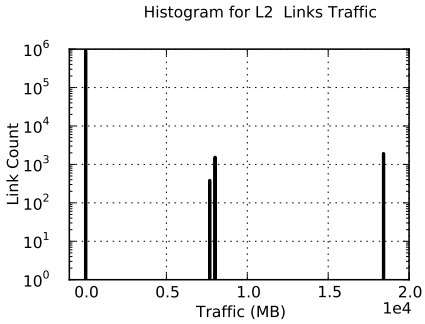

Fig. 6: 4D Stencil: distribution of traffic on L2 links for RDG.

Effect of Adaptivity: Use of AI further decreases the variation in traffic distribution. For many job placements (RDG, RRN, RRR), use of AI lowers the maximum traffic by up to 25%. Similar to UMesh, this gain is based on a better distribution of traffic on L1 links which leads to reduced maximum traffic. The adaptive hybrid routing provides a distribution that is similar to AI, but is marginally skewed by use of direct routes.

D. Many to Many Pattern (M2M)

In this pattern, the MPI processes are arranged in a 3-D grid with subsets being created along the Y axis. Within subsets of size 128, an all-to-all operation is performed. Such a pattern is representative of applications that perform many parallel Fast Fourier transform, e.g. pF3D [25], PARATEC, NAMD [26], and VASP [27]. Using the configuration presented in Table I, an MPI process's communicating partners are separated by multiples of 384 , i.e. a process *r* typically communicates with MPI ranks such as *r+384, r-384, r+2*384, r-2*384* etc. Depending on the position of a process in the 3D grid of the processes, the number of partners that are to the left and to the right of an MPI process varies. Also, as was the case with 4D Stencil, each MPI process in a set of consecutive MPI processes typically communicates with the corresponding MPI process in another set if the two sets are 384 ranks apart on an average.

Effect of Job Placement: Figure 7 shows the prediction results for M2M. In a manner similar to 4D Stencil, while the average and the median decreases on increasing the blocking for direct routing, albeit in lower proportions, the maximum traffic increases significantly. This increase is attributed to the overloading of certain L2 links as shown by the huge difference between the third quartile and the maximum in Figure 7 (bottom). This skewness is due to the non-uniform spread of communicating pairs described in the previous paragraph.

Effect of Indirect Routing: Use of indirect routing helps in offloading the overloaded L2 links, but it increases the load on L1 links (Figure 7 (bottom)). The extra load on L1 links is expected since indirect routing doubles the number of hops on an average. However, unlike the benchmarks we have seen so far, the maximum traffic is lower for direct routing with randomized placements and minimal blocking (RDN and RDR). We hypothesize that this is induced by a good distribution of traffic on links by randomized placement. The lower nearby values of the minimum, the median, and the quartiles for direct routing with randomized placement confirms this hypothesis. As a result, for M2M, direct routing is more likely to provide higher network throughput. We believe that such a distribution was not obtained for UMesh and 4D Stencil because of the fewer number of communicating partners with better MPI rank locality.

Effect of Adaptivity: The adaptive versions of the static routings had a positive but limited impact on the distribution of traffic. This is in part due to the limited opportunity available for adaptivity in already uniform distribution (for randomized placements and indirect routing). For cases with skewed distribution, e.g. SD with RRN, the skewness is caused by a few L2 links that are the only path available for the messages to traverse from one group to other (Figure 7 (bottom)). As a result, adaptivity cannot improve the distribution. The adaptive hybrid yields a distribution that resembles AI, but unlike earlier, use of direct routes helps it improve upon AI.

E. Random Neighbors Pattern (Spread)

This pattern spreads the communication uniformly in the system by making each MPI process communicate with $6-20$ neighbors selected randomly. In applications that perform computation aware load balancing, e.g. NAMD, or are not executed on near-by physical cores, such communication pattern arise. Figure 5 (bottom) shows the expected distribution of traffic for execution of Spread on the full system.

The first thing to observe is that almost all links are utilized irrespective of the job placement and the routing. This is a direct impact of the spread of the communicating pairs that the benchmark provides. Another effect of the spread is the minimal impact of the job placement on the load distribution. Next, we note that while the average quality of the distribution has improved, the gap between the maximum and other data points (average, median and quartiles) has increase significantly for indirect routings. Similar observation can be made for direct routing with randomized placement if we compare with the results for M2M. Further analysis of L1 and L2 links traffic distribution shows that such a skewness is caused by overloading of certain L1 links. We believe this is caused by non-uniformity in the communication pattern — randomization of communication patterns is probably not uniformly distributing them.

The next important observation from the Figure 5 (bottom) is the lower values of all data points (minimum, quartiles, average, and maximum) for direct routing in comparison to the indirect routing. This result is similar to what we described in M2M — given a sufficiently distributed communication pattern, indirect routing only adds extra traffic because of the extra hops it takes. Finally, we note that the adaptive versions of the routings reduce the maximum traffic by up to 10%. Other than that, they provide a very similar distribution. As we saw in M2M, the AH routing provides a distribution similar to AI with lower maximum traffic due to use of direct routes.

F. Summary of Full System Predictions

Based on the analysis so far, we list the following summarizing points for single jobs executed on full systems:

— For patterns with many communicating nearby MPI processes, blocking may reduce the average and quartiles (UMesh).

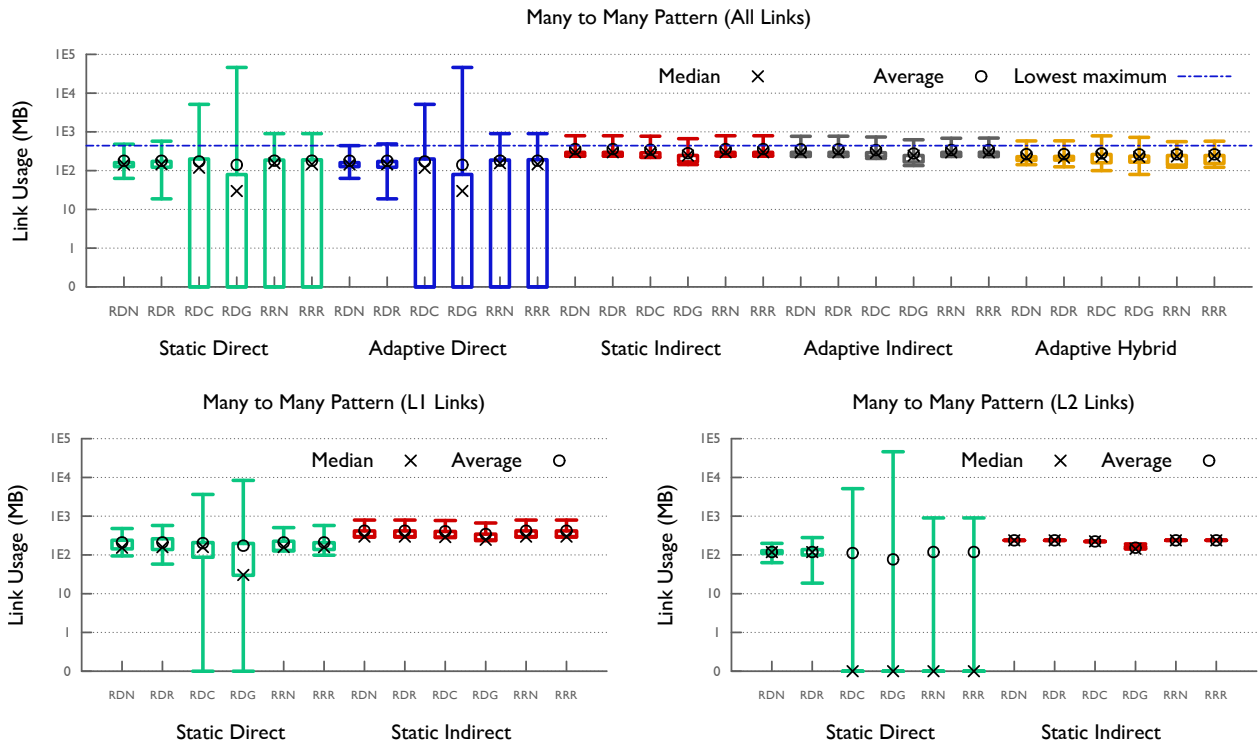

Fig. 7: Many to many pattern (M2M): direct routing with randomized placement has lower average and maximum traffic.

— Direct routing may overload a few links, especially L2 links, if the communication is distribute evenly (4D Stencil, M2M).
— Randomized placement spreads traffic for patterns with non-uniform distribution of traffic (4D Stencil, M2M).
— Indirect routing is helpful in improving the distribution of traffic, but typically increases the average traffic (all patterns).
— If the communication pattern and job placement spreads the communication uniformly, indirect routing may increase the quartiles and the maximum traffic (M2M, Spread).
— Adaptive routing typically provides a similar traffic distribution, but may lower the maximum traffic significantly. Thus, in order to save space, we avoid showing results for static routings in the rest of the paper.
— Adaptive hybrid provides a traffic distribution similar to AI, but may provide a higher or lower maximum traffic depending on the relative performance of AD and AI.

G. Variations in Job Size

We now present a case study in which one of the patterns, M2M, is executed in isolation on the full system, but occupies only a fraction of the cores. For comparison, we use M2M predictions on the full system from Figure 7 (top) and traffic distributions presented in Figure 8 for predictions using 66% and 33% of cores in isolation.

We observe very similar trends in traffic distribution across job placements and routings as we move from predictions for 100% cores to predictions for 33% cores. As expected, the absolute values of most data points (maximum, average, quartiles) decrease steadily for the combinations that provide a good distribution. Direct routing with randomized placements consistently outperform indirect routings for critical data points including the maximum traffic.

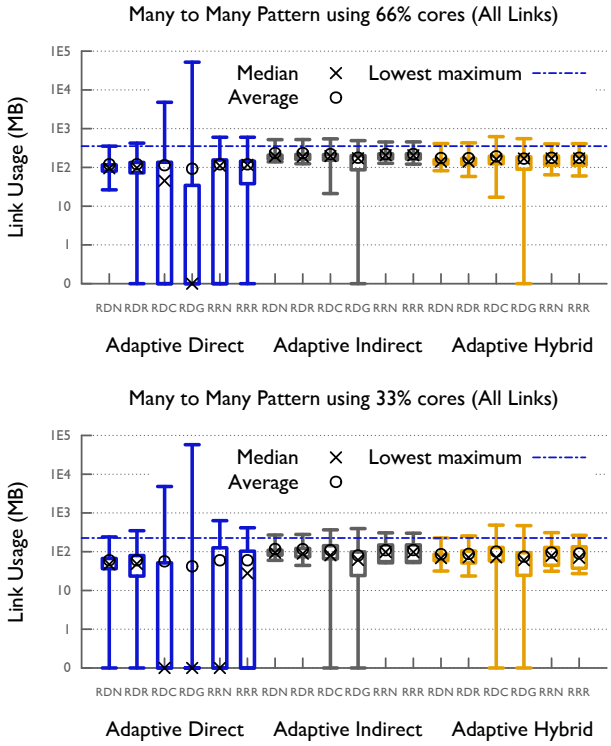

Fig. 8: Traffic distribution for M2M on 66% and 33% cores.

TABLE II: Percentage cores allocated to patterns in workloads.

Comm Pattern	Workload 1	Workload 2	Workload 3	Workload 4
UMesh	20	10	20	40
2D Stencil	10	10	40	10
4D Stencil	40	20	10	20
M2M	20	40	10	20
Spread	10	20	20	10

Benefits of adaptive routing are significantly higher for job executions with smaller core counts. For the 100%, 66% and 33% cores executions, adaptive routing reduces the maximum traffic by up to 10.2%, 31.1% and 35% respectively. We attribute the increasing effect of the adaptivity to the non-uniformity that use of a fraction of cores in the system induces. Adaptive routing is able to observe these non-uniformities, and guides the traffic towards a better distribution.

Finally, we draw attention to the adaptive hybrid routing. For job placements that suit AD for this pattern (RDN and RDR), as we move from 100% to 33% cores, the critical data points (maximum, average, median) for AH are significantly lesser than those for AI. In fact, for the 33% cores case, the maximum traffic is least for AH among all the routings. This suggests that as non-uniformity in the system increases, AH is able to judiciously capitalize on good attributes of both AD and AI — use direct routes when they are not congested, else use indirect routes to offload traffic.

VI. Predictions for Parallel Workloads

In this section, we focus on the more practical scenario in which multiple jobs with different patterns use the network simultaneously. Table II presents the representative workloads that we use for the experiments. These workloads represent capability jobs that use at least 10% of the total system size. For each workload, the system is divided among 5 single jobs that represent the following communication patterns: UMesh, 2D Stencil, 4D Stencil, M2M, and Spread. While four of these patterns are the ones described in Section IV, 2D Stencil is a new addition. It represents a two-dimensional stencil-based communication found in many applications such as WRF [28].

A. Comparing Different Parallel Workloads

Figure 9 presents the predicted traffic distribution for workloads listed in Table II. A common observation for all the workloads is the very high value for maximum traffic for AD with heavy blocking (RDC and RDG). Detailed histogram for the traffic on the links revealed that a few L2 links are heavily loaded. Initially, we suspected this to be caused by overloading of a few L2 links by 4D Stencil in a similar manner as we saw in Section V-C. In order to verify our assumption, we tried another workload with only four jobs: UMesh, Spread, M2M and 2D Stencil. However, for this fifth workload too, we observed similar overloading for AD with heavy blocking. Hence, we conclude that job placements with heavy blocking exposes any locality in communicating pairs of MPI ranks and leads to a few overloaded L2 links.

Figure 9 (a) presents the predicted traffic distribution for Workload 1, in which 40% of the cores are allocated to 4D Stencil; UMesh and M2M are assigned 20% cores each. For AD with blocked placement (RDC and RDG), we note that the average traffic is significantly higher than the median — a characteristic of 4D Stencil which occupies 40% of the cores in this workload. Use of randomized placement and indirect routing helps in reducing the skewness and maximum traffic. Among the combinations with similar distributions, the maximum traffic is lowest for AI with RRR placement and AH with RDN/RDR placement. Adaptive routings reduce the maximum traffic by up to 35% in comparison to their static counterparts.

In Workload 2, M2M is allocated the most number of cores (40%), while 4D Stencil and Spread are allocated 20% cores each. Other than the impact of locality in communicating pairs for AD with RDC and RDG described earlier, one can observe the impact of higher fraction of Spread and M2M in the closer values for average, median, and the quartiles. It also leads to AD with RRR and AH with RDN/RDR having the lowest value for the maximum traffic. Similar to Workload 1, adaptivity reduces the maximum traffic by up to 34.3%.

2D Stencil is assigned the largest number of cores (40%) in Workload 3, with UMesh and Spread being allocated 20% cores each. In 2D Stencil, four messages of size 64 KB are exchanged with its neighbors. For Workload 3, the traffic distribution shows mixed impact of Spread and 2D Stencil in Figure 9 (c). Contribution from Spread leads to a general increase in the maximum traffic for AI, while the gains obtained by randomized placements of 2D Stencil lower the maximum traffic for those combinations. Overall, the AH routing appears to take advantage of these effects and provides a nice distribution with the lowest value of maximum traffic for RDN and RDR. For Workload 4, predictions shown in Figure 9 (d) are very similar to Workload 3.

We make the following conclusions from these results: 1) Single capability jobs may have a significant impact on the traffic distribution of a workload, especially on its skewness as shown by the impact of 4D Stencil, 2) Similar traffic distributions are observed for workloads with the same set of jobs executing in different proportions, 3) The adaptive hybrid routing is able to combine positive features of AD and AI, thus providing a better traffic distribution.

B. Job-specific Routing

Results presented in this section are for another interesting scenario in which each job in a workload is allowed to use a routing of its choice. This is currently not allowed on most systems but might become a useful option as system sizes increase further. We use Workload 2 and Workload 4 from Table II for these experiments. For each job, we select the routing that resulted in the lowest maximum traffic for a given job placement when the job was run by itself (Section V).

Comparison of the traffic distribution for Workload 2, shown in Figure 10, with the results in Figure 9 (b) indicates that the distribution for job-specific routing is most similar to that of AH. However, for certain job placements, e.g. RDN and RDR, it has lower values for minimum traffic and first quartiles — a characteristic shown by AD routing for Workload 2. This is not surprising because Workload 2 is dominated by M2M and Spread for which AD and AH were the best routings.

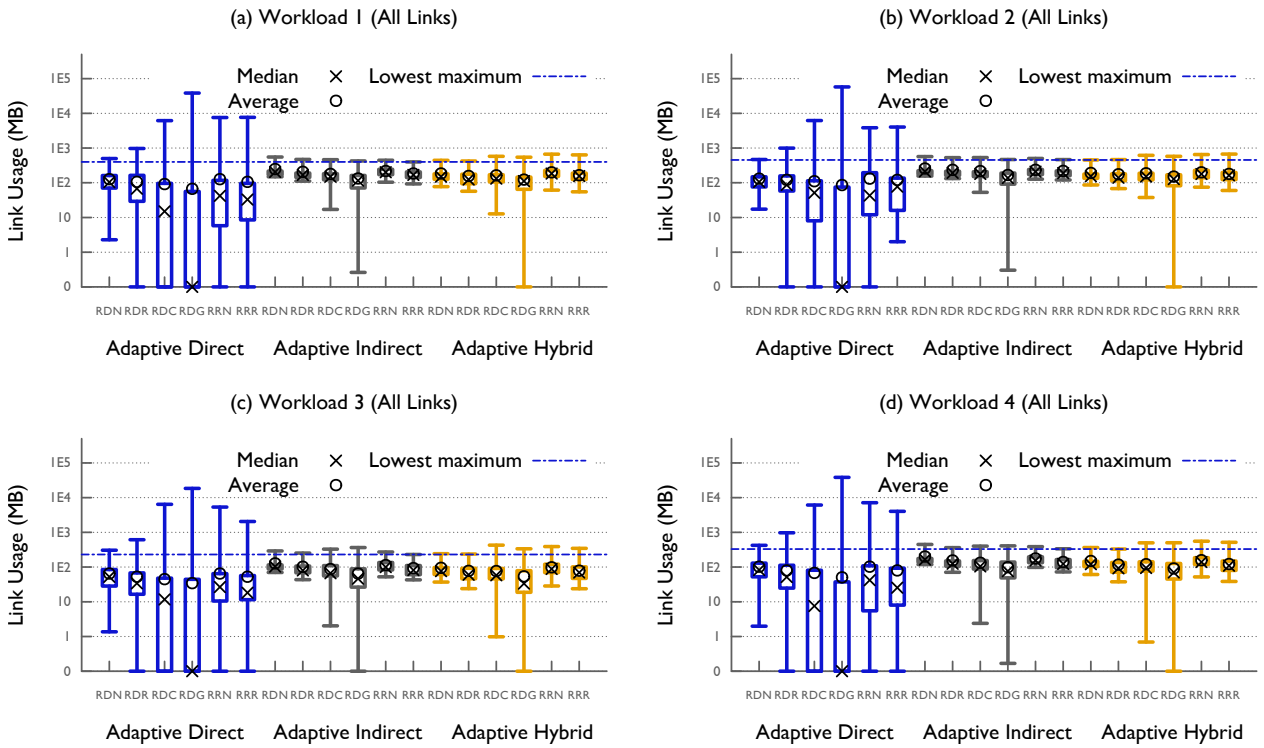

Fig. 9: Parallel workloads traffic distribution.

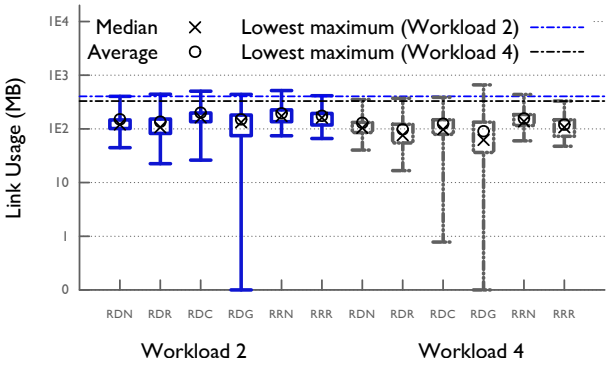

Fig. 10: Job-specific routing traffic distribution (All Links).

An important observation to make is that the use of job-specific routing reduces the maximum traffic on any link for all job placements. Similarly, for Workload 4, the distribution of traffic for job-specific routing is similar to the load distribution for AI (Figure 9 (d)) which was the best performing routing for UMesh and 4D Stencil that dominate it. It also provides similar maximum traffic for best performing job placements.

VII. Conclusion

In this paper, we presented a comparative analysis of various routing strategies and job placement policies w.r.t. network link throughput for the dragonfly topology. We have developed a congestion-aware model to determine the traffic distribution given a communication trace and a routing strategy. The output of this model is used to answer the questions we posed in the introduction. The answer to the first question is more nuanced than the other two because it depends heavily on the application communication patterns. The general observations are that a randomized placement at the granularity of nodes and routers and/or indirect routing can help spread the messaging traffic over the network and reduce hot-spots. If the communication pattern results in non-uniform distribution of traffic, adaptive routing may provide significantly better traffic distributions by reducing hot-spots.

For parallel job workloads (second question), adaptive hybrid routing is useful for combining good features of adaptive direct and adaptive indirect routings and may provide a good traffic distribution with lower maximum traffic. Adaptive routings also improve the traffic distribution significantly in comparison to static routings. We also observed that allowing the users to choose a routing for their application can be beneficial in most cases on dragonfly networks (third question). Use of randomized placement at the granularity of nodes and routers is the suggested choice for such scenarios also. We believe that the model developed in this paper will enable system administrators and application end-users to try different scenarios and help them optimize network throughput for their use-cases.

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. This work was funded by the Laboratory Directed Research and Development Program at LLNL under project tracking code 13-ERD-055 (LLNL-CONF-653557).

Experiments for this work were performed on Mira and

Vesta, IBM Blue Gene/Q installations at Argonne National Laboratory. The authors would like to acknowledge PEACEndStation and PARTS projects for the machine allocations provided by them. The authors would also like to acknowledge the staff of Pittsburgh Supercomputing Center and XSEDE for allocation on Blacklight.

REFERENCES

[1] 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.

[2] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick, "Exascale computing study: Technology challenges in achieving exascale systems," 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 *2010 IEEE 18th Annual Symposium on High Performance Interconnects (HOTI)*, August 2010, pp. 75–82.

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

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

[6] K. Antypas, J. Shalf, and H. Wasserman, "NERSC6 Workload Analysis and Benchmark Selection Process," Lawrence Berkeley National Lab, Tech. Rep. LBNL-1014E, 2008.

[7] B. Austin, M. Cordery, H. Wasserman, and N. Wright, "Performance measurements of the nersc cray cascade system." Cray, Inc., May 2013.

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

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

[10] M. I. Frank, A. Agarwal, and M. K. Vernon, "Lopc: modeling contention in parallel algorithms," in *Proceedings of the sixth ACM SIGPLAN symposium on Principles and practice of parallel programming*, ser. PPOPP '97. New York, NY, USA: ACM, 1997, pp. 276–287. [Online]. Available: http://doi.acm.org/10.1145/263764.263803

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

[12] C. Moritz and M. Frank, "Logpg: Modeling network contention in message-passing programs," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 12, no. 4, pp. 404 –415, apr 2001.

[13] W. Chen, J. Zhai, J. Zhang, and W. Zheng, "Loggpo: An accurate communication model for performance prediction of mpi programs," *Science in China Series F: Information Sciences*, vol. 52, no. 10, pp. 1785–1791, 2009.

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

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

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

[17] G. Zheng, G. Kakulapati, and L. V. Kalé, "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, April 2004, p. 78.

[18] V. T. Chakaravarthy, N. P. K. Katta, M. Kedia, Y. Sabharwal, A. Ramanan, and R. Rajamony, "Mapping Strategies for the PERCS Architecture," in *19th annual IEEE International Conference on High Performance Computing (HiPC 2012)*, December 2012.

[19] N. Jain, A. Bhatele, M. P. Robson, T. Gamblin, and L. V. Kale, "Predicting application performance using supervised learning on communication features," in *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. IEEE Computer Society, Nov. 2013, lLNL-CONF-635857.

[20] B. Prisacari, G. Rodriguez, P. Heidelberger, D. Chen, C. Minkenberg, and T. Hoefler, "Efficient task placement and routing of nearest neighbor exchanges in dragonfly networks," in *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, ser. HPDC '14. ACM, 2014, pp. 129–140. [Online]. Available: http://doi.acm.org/10.1145/2600212.2600225

[21] S. Kamil, L. Oliker, A. Pinar, and J. Shalf, "Communication requirements and interconnect optimization for high-end scientific applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 2, pp. 188–202, Feb. 2010.

[22] C. Huang, O. Lawlor, and L. V. Kalé, "Adaptive MPI," in *Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2003), LNCS 2958*, College Station, Texas, October 2003, pp. 306–322.

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

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

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

[26] J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kalé, "NAMD: Biomolecular simulation on thousands of processors," in *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, Baltimore, MD, September 2002, pp. 1–18.

[27] G. Kresse and J. Hafner, "Ab initio molecular dynamics for liquid metals," *Phys. Rev. B*, vol. 47, p. 558, 1993.

[28] W. C. Skamarock, J. B. Klemp, J. Dudhia, D. O. Gill, D. M. Barker, W. Wang, and J. G. Powers, "A description of the advanced research wrf version 2," NCAR, Tech. Rep. Technical Note NCAR/TN-468+STR, June 2005.