Level-Spread: A New Job Allocation Policy for Dragonfly Networks

Yijia Zhang*, Ozan Tuncer*, Fulya Kaplan*, Katzalin Olcoz†, Vitus J. Leung‡ and Ayse K. Coskun*
* Boston University, Boston, MA 02215, USA; E-mail: {zhangyj, otuncer, fkaplan3, acoskun}@bu.edu
† Universidad Complutense de Madrid, Madrid 28040, Spain; E-mail: katzalin@ucm.es
‡ Sandia National Laboratories, Albuquerque, NM 87185, USA; E-mail: vjleung@sandia.gov

***Abstract*—The *dragonfly* network topology has attracted attention in recent years owing to its high radix and constant diameter. However, the influence of job allocation on communication time in dragonfly networks is not fully understood. Recent studies have shown that random allocation is better at balancing the network traffic, while compact allocation is better at harnessing the locality in dragonfly groups. Based on these observations, this paper introduces a novel allocation policy called *Level-Spread* for dragonfly networks. This policy spreads jobs within the smallest network level that a given job can fit in at the time of its allocation. In this way, it simultaneously harnesses node adjacency and balances link congestion. To evaluate the performance of *Level-Spread*, we run packet-level network simulations using a diverse set of application communication patterns, job sizes, and communication intensities. We also explore the impact of network properties such as the number of groups, number of routers per group, machine utilization level, and global link bandwidth. *Level-Spread* reduces the communication overhead by 16% on average (and up to 71%) compared to the state-of-the-art allocation policies.**

I. Introduction

Efficient system management in ever-growing high performance computing (HPC) systems is a common concern of designers, administrators, and users. As the number of cores required by parallel programs continues to increase, network communication time among the compute nodes becomes a performance bottleneck [1], [2]. Thus, reducing the communication cost on HPC systems is essential for better utilization of valuable computation resources.

Dragonfly [3] is a network topology utilizing high-radix routers, and has attracted attention in recent years [4]–[10]. Dragonfly networks are composed of interconnected *groups* that behave as virtual routers. A group contains a set of compute nodes, network links, and routers. One example of a group is shown in Fig. 1. Different groups are connected by optical *global links*, and routers in each group are connected by electrical *local links*. Each router is connected to several nodes. Owing to the all-to-all connections among its groups, dragonfly is a constant-diameter network (i.e., the diameter does not scale up with the increase in the number of nodes). A message sent from one node to another following the shortest-path route passes through at most two local links and one global link. The dragonfly topology has been implemented in some of the latest HPC systems [11], [12].

An important factor that affects the communication time in HPC systems is job allocation [2], which is the procedure of selecting a set of compute nodes to run a parallel program. The hierarchy of the dragonfly topology brings new complexity to job allocation, and to date, job allocation on dragonfly has not been fully explored. Most studies on dragonfly networks only consider two allocation policies: *simple* allocation (i.e., selecting nodes by following the label order) and *random* allocation [11]. Jain et al. [13] proposed six different job allocation policies for dragonfly machines and concluded that random allocation is generally better than the others. Yang et al. [9] further demonstrated that random allocation cannot guarantee the best performance for every job.

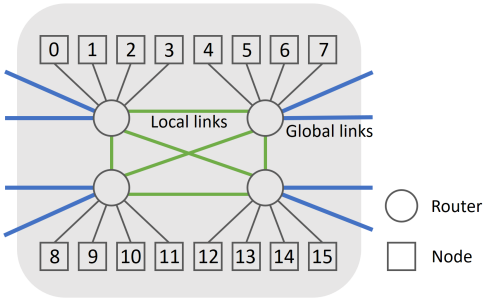


Fig. 1: A group in a dragonfly network. The group is composed of 4 routers that are connected to each other by local links (green). Each router is connected to 4 nodes, which are labeled by numbers. The entire dragonfly machine (not shown here) consists of a number of such groups. Between each pair of groups there is a global link (blue) connecting them together.

Inspired by the prior work, we examine the shortcomings of existing allocation policies and conclude that a size-aware job allocation policy is necessary to further improve the performance on dragonfly networks. To this end, we propose the *Level-Spread allocation policy* (Sec. III) and compare it with several state-of-the-art allocation policies (Sec. IV). We perform extensive simulations (Sec. V) and show that our proposed policy outperforms the existing policies (Sec. VI). In addition, we discuss the influence of scheduling on the performance of allocation policies (Sec. VI-E). The specific contributions of our work are as follows:

- We demonstrate that on dragonfly networks, harnessing node locality and balancing link congestion are two complimentary approaches to reduce communication latency. Existing allocation policies only resemble either one of these two approaches. We show that combining them intelligently provides better performance.

- We design a novel allocation policy, *Level-Spread*, for dragonfly networks. This policy spreads jobs within the smallest network level that a given job can fit in. Through packet-level network simulations, we show that our proposed policy reduces the communication time by 16% on average compared to the state-of-the-art policies by harnessing node locality and balancing link congestion.

II. Related Work

Prior work observed the impact of job interference on the performance of HPC systems [1], [2], [14], [15]. Bhatele et al. [1] and Leung et al. [2] demonstrated the performance variation of parallel programs caused by job allocation and communication interference on a torus network.

Several studies explored the influence of job allocation on dragonfly networks. Jain et al. [13] compared the performance of six dragonfly-specific job allocation policies. They observed that random allocation is generally beneficial in spreading network traffic and reducing communication hot spots. We include these allocation policies as part of the baseline policies in our work (Sec. IV). Budiardja et al. [16] showed that spreading jobs to all groups during allocation distributes the network traffic, and thus, reduces congestion.

Yang et al. [9] observed performance degradation due to *random* allocation when multiple jobs run simultaneously on a dragonfly machine. They found that the network congestion caused by communication-intensive jobs greatly impacts the performance of other jobs. They further anticipated a combined approach of *simple* and *random* allocation. However, they did not propose a specific allocation policy.

Job allocation on traditional network topologies such as fat-tree also inspired our work. For example, Jokanovic et al. [14] proposed a size-aware policy that alleviates communication interference by allocating large jobs on one side of the system and small jobs on the other side.

On the topic of link configuration and bandwidth for dragonfly machines, Groves et al. [8] analyzed the influence of link bandwidth on job execution times. Several studies compared different global link arrangements for dragonfly networks in terms of theoretical bisection bandwidth [4], [17]. Routing algorithms for dragonfly networks have also been explored extensively [3], [6], [13], [18], [19]. Task mapping, which refers to mapping the tasks of a parallel application onto the processors of the allocated computing nodes, has also been studied on dragonfly networks [20], [21].

Our work distinguishes from the related work by the following points. First, we propose a size-aware job allocation policy, *Level-Spread*, specifically for the dragonfly topology. Prior work on size-aware allocation policies either focus on other network topologies or demonstrate performance tradeoffs without providing a concrete policy. Second, we evaluate our allocation policy on a broad range of workloads, machine configurations, and communication characteristics, instead of focusing on a specific job type or network setting.

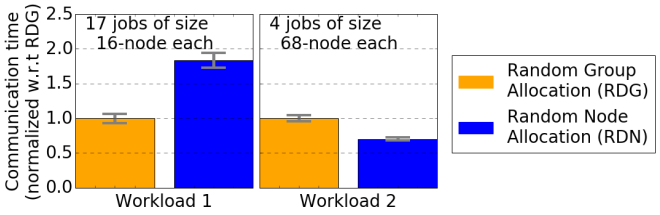


Fig. 2: We compare random group allocation (RDG), which prioritizes selecting nodes from the same group when allocating a parallel job, and random node allocation (RDN), which selects nodes randomly across the entire network, and we simulate two different workloads. Workload 1 (small jobs) benefits from RDG, whereas workload 2 (large jobs) benefits from RDN.

III. Level-spread allocation policy

This section describes our *Level-Spread* policy. In Sec. III-A, we show that the best-performing allocation strategy is different for jobs of different sizes, and we explain why spreading tasks of a job inside a suitable network level can be beneficial to their MPI (Message Passing Interface) communication. We then present our *Level-Spread* in Sec. III-B.

A. Motivation

Intuitively, allocating jobs compactly reduces communication times as it leads to small network distances. However, in dragonflies, allocating large jobs compactly (i.e., prioritizing using nodes from the same group when allocating a job) leads to hot spots on network links. Hence, several studies suggest *random* allocation for dragonfly networks [9], [13], [22].

Our simulation results in Fig. 2 confirm this phenomenon. The figure compares the communication time by running two workloads on a 272-node dragonfly machine with 16 nodes per group. With each workload, we compare two allocation policies proposed by Jain et al. [13]: (1) random group allocation (RDG), which randomly selects a group and allocates the job to all the idle nodes in that group, and repeats this process if more nodes are required; (2) random node allocation (RDN), which randomly selects nodes in the entire machine for a job. Each simulation is further repeated 10 times to reduce the bias due to randomness.

In workload 1, seventeen 16-node jobs are running simultaneously, fully utilizing the dragonfly machine. RDG outperforms RDN as RDG allocates each job to a single group, harnessing group-level locality and avoiding interference on global links. Contrary to workload 1, in workload 2, where four 68-node jobs are running simultaneously on the same machine, RDN reduces the communication time by 30% compared to RDG. Although more packets travel on global links with RDN than with RDG, the RDN policy benefits from a balanced load on the network.

These observations inspire us to design a new allocation policy combining the advantages of both RDG and RDN.

B. Level-Spread allocation policy

Our goal is to minimize the communication time of jobs running on HPC systems with dragonfly topologies through

job allocation. Here, a job consists of multiple parallel tasks, and each task is defined as an MPI rank. In contrast to existing dragonfly-specific allocation policies that apply the same allocation strategy (grouping or spreading) for all jobs [13], our *Level-Spread policy* selects specific allocation strategies based on the job size along with the machine state.

The principle of our policy is to allocate a job into the smallest network level (router, group, or machine) where that job can fit in and to spread the parallel tasks of the job within that level. Selecting the smallest network level benefits a job by letting it harness node locality and by reducing the communication interference with other jobs. Spreading the tasks within that level balances the network communication on the links, and thus, reduces network hot spots. The specific steps are as follows:

- If a job fits within the available nodes that are connected to a single router, we select *the router with the largest number of idle nodes* and allocate the job there.
- If a job cannot fit within a single router but fits within the available nodes in a single group, we select *the most idle group* and allocate the job there. To further reduce load imbalance on local links in this group, we select nodes connected to different routers in a round-robin manner.
- If a job cannot fit within a single group, we *spread the job throughout the entire network*, where we select nodes in different groups in a round-robin manner.

Following the terminology used by Kim *et al.* [3], in this article, p represents the number of nodes connected to a router; a represents the number of routers in a group; g represents the number of groups in the entire dragonfly machine.

Fig. 3 illustrates an example allocation for three jobs. The first 4-node job is allocated to a single router. The second 8-node job (green) is spread within the group with the largest number of idle nodes. The third job (blue) is spread throughout the entire machine, occupying the first six available nodes of every group.

In the implementation of our algorithm, we scan through all routers/groups when searching for the router/group with the maximum availability. The time complexity of our algorithm is linear with the number of nodes in the machine, i.e., O($a \times g \times p$), as follows: Allocating jobs that fit to a single router, is performed in O($a \times g + p$) time, where selecting a router is O($a \times g$) and allocating nodes within the router is O(p). The allocation of the jobs that do not fit in a single router but do fit within a group is performed in O($a \times g + a \times p$). For the remaining jobs, the algorithm simply scans through all nodes in O($a \times g \times p$). Owing to its linear time complexity, our policy can be easily implemented on real dragonfly machines.

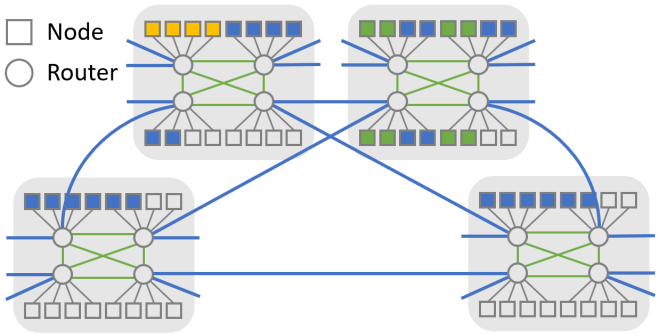


Fig. 3: Three jobs are allocated to a dragonfly machine ($g = 9$, $a = 4$, $p = 4$) by *Level-Spread*. Only four groups are drawn for simplicity. The first job (orange) is allocated to the nodes connected to a single router. The second job (green) is allocated to different routers in the same group. The third job (blue) is spread to all groups in a round-robin manner.

IV. Baseline allocation policies

We select the following seven allocation policies [13], [23] for dragonfly networks as baselines for comparison:

- **Simple** selects idle nodes by following the node label order, which is defined as follows: the nodes in the first group are labeled as in Figure 1 and the labeling continues with the next group in the network following the same rationale. Simple policy allocates jobs in a compact manner in general.
- **Slurm** allocates jobs following the policy for dragonfly implemented in the Slurm Workload Manager [23]. It first attempts to allocate a job to the nodes connected to a single router. The first available router with a sufficient number of idle nodes is chosen, and the nodes connected to that router are allocated following the label order. If there are no routers with a sufficient number of idle nodes, it searches for the router with the fewest number of idle nodes and selects the idle nodes connected to that router following the label order, and repeats this step as necessary. This allocation policy does not utilize the group structure of a dragonfly network.
- **Random Nodes (RDN)** chooses nodes completely randomly from the entire machine.
- **Random Routers (RDR)** randomly selects a router and then selects idle nodes connected to that router following the label order, and repeats this step as necessary.
- **Random Group (RDG)** randomly chooses a group and selects the nodes in that group following the label order, and repeats this step as necessary.
- **Round Robin Nodes (RRN)** starts from the first group, selects the first idle node following the label order in that group, then moves to the next group and repeats the same process as necessary.
- **Round Robin Routers (RRR)** starts from the first group, chooses the first available router following the label order and selects the idle nodes connected to that router following the label order. It then moves to the next group and repeats the same process as necessary.

We also compare our policy with a job-size-aware policy for fat-tree networks, proposed by Jokanovic et al. [14], and adapt their policy to dragonflies. Their policy places a virtual boundary dividing the machine into two partitions. When allocating a job whose size is smaller than the number of nodes per group, the policy allocates the job to a group in the first partition that has enough idle nodes. When allocating

TABLE I: Configurations of simulated dragonfly machines.

Machine Parameters	Values
Processors per node	2
Nodes per router (p)	4
Routers per group (a)	4 or 8
Number of groups (g)	17 or 33
Total number of nodes ($p \times a \times g$)	From 272 to 1056
Local link bandwidth	8 Gbit/s
Global link bandwidth	2 Gbit/s, 8 Gbit/s, or 32 Gbit/s
Routing algorithm	Adaptive routing
Machine utilization level	90% or 70%

larger jobs, the policy places the job in the second partition, starting from the last nodes (according to the label order). In this way, this policy reduces system fragmentation as well as interference between small and large jobs. The boundary between two partitions is updated dynamically based on the allocation history. Results for Jokanovic's policy are discussed in Sec. VI-F.

Our *Level-Spread* policy distinguishes from the existing policies by combining the grouping and spreading allocation strategy intelligently based on job sizes. Existing policies only relies on either the grouping or the spreading strategy, and cannot take full advantage of the specific hierarchical structure of dragonflies.

V. Experimental methodology

A commonly-used method to examine the performance of dragonfly networks and to explore various network configurations is to run network simulations [7], [9], [13], [20], [24]–[26]. To evaluate the performance of different job allocation policies on dragonflies, we run network simulations using the Structural Simulation Toolkit (SST) [27] with different network configurations and job communication patterns.

A. Structural Simulation Toolkit (SST)

The SST [27] is an open-source architectural framework developed to model and simulate HPC systems. It supports packet-level network simulations and has been verified and used in recent studies [8], [27]–[30]. We have extended SST by adding new allocation policies for the dragonfly network: the baseline policies and our *Level-Spread* policy.

B. Simulated Environments

We simulate dragonfly networks with various configurations, listed in Table I. We select these parameters following previous studies [9], [10], [20]. Our dragonfly machines have 16 or 32 ($= p \times a$) nodes per group. Thus, *Level-Spread* will spread jobs that are larger than 16 or 32 nodes to the entire machine. Following the designed structure of dragonfly network in Ref. [3], the routers inside each group are connected to each other by local links in an all-to-all fashion.

For message routing, we use the *adaptive routing* algorithm [3], which has been shown to provide good performance in dragonfly networks [3], [9], [13]. Based on link congestions derived from the local queue information, *adaptive routing* dynamically chooses between shortest-path routing and *Valiant routing*, which first directs each packet to a randomly-selected intermediate group and then to the destination group.

TABLE II: Parallel workloads in our experiments.

Type	Workload
Homogeneous workloads	Multiple 16-node jobs
	Multiple 64-node jobs
	Four quarter-machine-size jobs
Heterogeneous workloads	16-node jobs and 64-node jobs
	Jobs with randomly-selected sizes

Different from job allocation, *task mapping* controls the order of task placement onto the processors of the allocated nodes. We use random task mapping to reduce the bias caused by the task mapping process. We assume that each node has two processors where each processor executes one task.

We also explore various machine utilization levels, defined as the number of busy nodes divided by the total number of nodes in the machine. As real HPC systems are heavily utilized, we test utilization levels of 90% and 70%.

C. Parallel Workloads

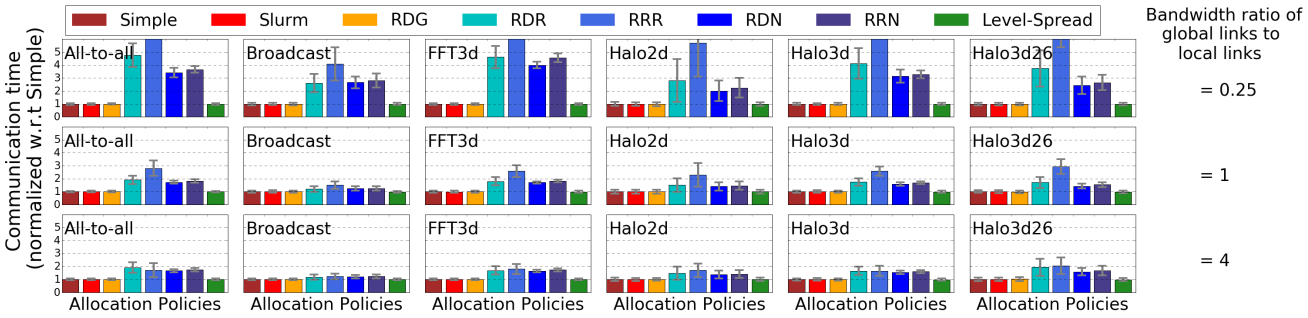
We examine both homogeneous and heterogeneous workloads listed in Table II. For each homogeneous workload, the number of jobs is determined by machine size and utilization. For example, for a 1056-node dragonfly machine and 90% utilization level, the homogeneous workload of 16-node jobs consists of $\lfloor 1056 \times 90\%/16 \rfloor = 59$ such jobs. For each heterogeneous workload, we set the number of 16-node jobs the same as the number of 64-node jobs. To explore a broader range of job sizes and numbers, we also generate random workloads each composed of two types of jobs with randomly-generated sizes (see Sec. VI-D).

All jobs in our workloads arrive at the same time. The order of jobs to be allocated, which is determined by scheduling algorithm, influences the outcome of allocation. As allocation is typically performed independent of scheduling, in Sec. VI-E we investigate two job ordering scenarios: small-job first and large-job first.

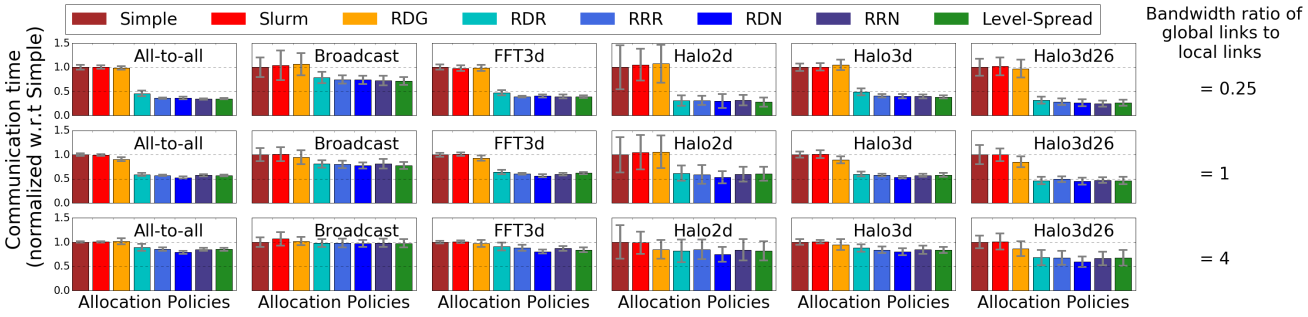
The communication structure among the tasks of a job defines the job's communication pattern. To explore the influence of job communication patterns on the performance, we use the following six communication patterns, which represent common communication characteristics in parallel HPC applications [31]–[35]:

- **All-to-all**: each task sends messages to all other tasks.
- **Broadcast**: one central task broadcasts messages to all other tasks.
- **FFT3d**: the communication pattern of doing 3-D fast Fourier transform.
- **Halo2d**: each task sends messages to its 4 nearest neighbors, forming a communication graph in a 2-D grid.
- **Halo3d**: each task sends messages to its 6 nearest neighbors, forming a communication graph in a 3-D grid.
- **Halo3d26**: each task sends messages to its 26 neighbors in 3 dimensions, including 6 nearest neighbors and 20 diagonal neighbors.

In order to focus on the communication overhead due to job allocation, we simulate jobs with only communication without



(a) Workload composed of 16-node jobs. As the jobs fit in a single group, this workload benefits from the grouping-strategy policies (*Level-Spread*, Simple, Slurm, RDG), which, in this case, allocate the tasks of a job into the same group.



(b) Workload composed of 64-node jobs. As the jobs do not fit in a single group, this workload benefits from spreading-strategy policies (*Level-Spread*, RDR, RRR, RDN, RRN), which, in this case, spread tasks of a job into different groups.

Fig. 4: Communication time of homogeneous workloads on a machine with 16 nodes per group, 17 groups in total, at 90% utilization, and using an application message size of 1 KB. Results are normalized with respect to the Simple allocation policy. Error bars represent the standard deviation. Both (a) and (b) shows reduced difference among these allocations when the global link bandwidth increases relative to the local link bandwidth.

any computation. We also explore the influence of communication intensity by using 1KB and 100KB job message sizes.

To reduce the influence of randomness introduced by task mapping, routing, and allocation policies with randomization, we repeat each experiment ten times.

We explore all the combinations of the machine configurations listed in Table I, workloads listed in Table II, and communication characteristics discussed above. In total, we conduct more than one hundred thousand simulations.

VI. Results and Discussions

In this section, we provide our experimental results and analyze the strengths and weaknesses of the baseline policies as well as our proposed policy. We first display the results from running homogeneous and heterogeneous workloads. Next, we analyze the impact of communication intensity on performance. In Sec. VI-D, we examine the performance on a broad range of random workloads. In Sec. VI-E, we discuss the influence of scheduling on the performance of *Level-Spread*. In Sec. VI-F, we compare *Level-Spread* with a size-aware allocation policy proposed by Jokanovic et al. for fat-tree networks.

We focus on communication time, which is a better metric than throughput to evaluate the network performance because it has a more direct impact on job execution time [20]. As users generally care about the relative degree of the delay of HPC jobs, we use normalized values in our evaluation.

A. Homogeneous Workloads

Figure 4 shows the average communication time of the homogeneous workloads. The results in each subfigure is normalized with respect to the average communication time using the Simple allocation policy. We simulate workloads that consist of the same communication pattern, for all six communication patterns described in Sec. V-C. Each column of subfigures focuses on a communication pattern, while each row focuses on a different ratio of global link bandwidth to local link bandwidth.

For the 16-node-job workloads in Fig. 4(a), because the smallest network level that a single job can fit in is a dragonfly group, *Level-Spread* allocates the tasks of each job within a single group. *Level-Spread* and policies that select nodes compactly (Simple, Slurm, and RDG) outperform the policies that spread the tasks across the entire machine (RDR, RRR, RDN, and RRN) by reducing the communication time by 15% to 89%. The degree of reduction depends significantly on link bandwidth.

Figure 4(b) shows the normalized communication time in workloads composed of 64-node jobs. As a 64-node job cannot fit in a single dragonfly group, *Level-Spread* spreads the tasks of each job across the entire machine. These results demonstrate that *Level-Spread* and the policies that spread the tasks (RDR, RRR, RDN, RRN) outperform the other policies that select nodes compactly. When the bandwidth ratio of global-to-local links is 1, which is close to the values in existing

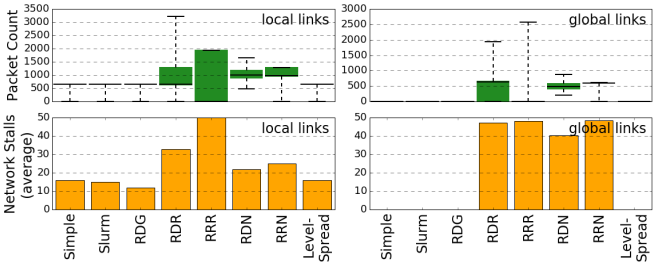


Fig. 5: Packet count and stalls at the output port to global or local links when running the 16-node-job homogeneous workload.

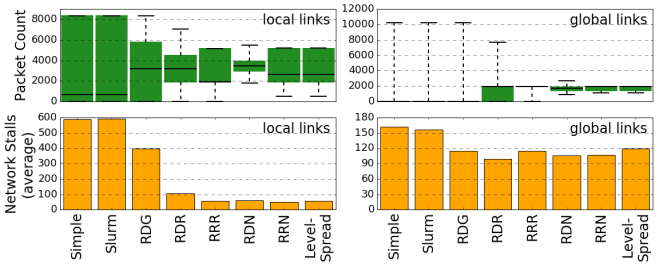


Fig. 6: Packet count and stalls at the output port to global or local links when running the 64-node-job homogeneous workload.

dragonfly machines, *Level-Spread* reduces the communication time by 22% to 54% compared to Simple allocation.

By comparing different columns in Fig. 4, we see that the performance of the eight allocation policies is consistent for the six communication patterns. By comparing different rows, we see that the increase of global link bandwidth reduces performance difference among allocations.

For all machine parameters and communication patterns we use, we have also examined the impact of machine utilization. For the two utilization levels, 90% and 70%, our results show negligible impact of utilization level on the relative performance of different allocation policies.

We collect network statistics in these simulations to understand the underlying causes of the performance difference. Fig. 5 shows the number of packets and stalls on network links when the machine is running the 16-node-job All-to-all workload. The green boxes represent 25% to 75% percentiles, the central line represents median, and the whiskers represent min/max. In this case, the global-to-local link bandwidth is 1, job message size is 1KB, and the machine has 17 groups and 16 nodes per group. Fig. 5 shows that *Level-Spread* and the grouping-strategy policies (Simple, Slurm, RDG) indeed stress the links less than other policies. Consequently, grouping-strategy policies lead to fewer stalls, leading to better performance.

Similarly, Fig. 6 shows network statistics when the machine is running the 64-node-job All-to-all workload. *Level-Spread* and the spreading-strategy policies (RDR, RRR, RDN, RRN) lead to more packets on both local and global links in terms of median. However, spreading-strategy policies create smaller variations on packet count, in agreement with our expectation that spreading-strategy policies balance load on the links, and thus, reduce hot spots. Therefore, spreading-strategy policies result in fewer stalls compared to grouping-strategy policies, which explains their better performance.

To examine the performance of running jobs larger than 64-nodes, we also ran homogeneous workloads that consist of four jobs, each of quarter-machine-size. For the four machines ranging from 272 nodes to 1056 nodes, the performance of different allocations on these workloads are very close to Fig. 4(b); so we omit these results due to space constraints.

B. Heterogeneous Workloads

We evaluate the benefits of size awareness in our *Level-Spread* allocation policy using heterogeneous workloads that consist of jobs with different sizes. Fig. 7 shows the communication time of jobs in heterogeneous workloads composed of n 16-node and n 64-node jobs, where n is determined by the machine size and the target utilization level. We simulate six such workloads, each using jobs from one of the six communication patterns. We allocate small jobs first; the impact of the allocation order is studied in Sec. VI-E.

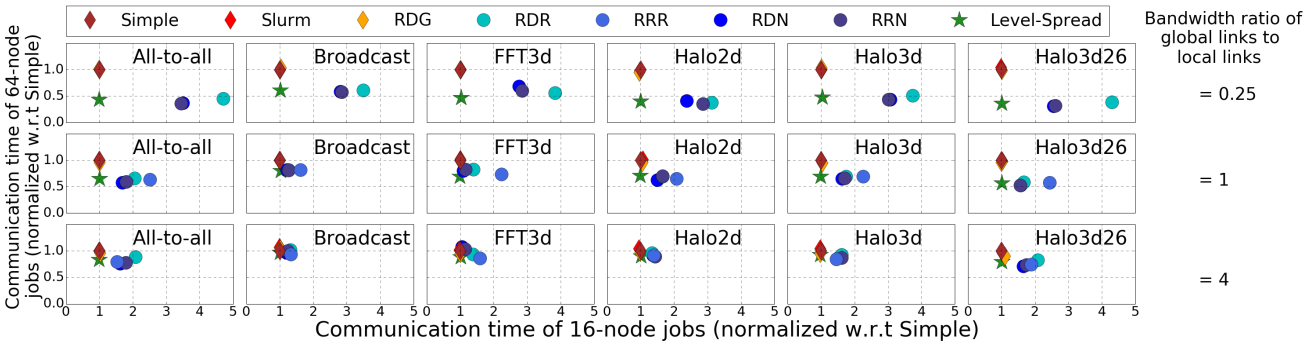
In every subfigure, the X-axis of a point represents the average communication time of the 16-node jobs, and the Y-axis represents that of the 64-node jobs. Values are normalized with respect to the Simple allocation policy in each subfigure. The grouping-strategy policies (Simple, Slurm, RDG) are marked with warm-colored diamonds, and the spreading-strategy policies (RDR, RRR, RDN, RRN) are marked with cool-colored circles. *Level-Spread* is marked with a green star.

All subfigures in Fig. 7 show that *Level-Spread* lies in the bottom-left part, which demonstrates that *Level-Spread* reduces the communication time for both small and large jobs. The X-axis of *Level-Spread* coincides with the X-axis of diamonds, showing that the communication time of the 16-node jobs allocated by *Level-Spread* remains similar to the Simple, Slurm, and RDG policies that allocate jobs compactly. Meanwhile, the Y-axis of *Level-Spread* coincides with the Y-axis of circles in general, showing that the communication time of the 64-node jobs allocated by the *Level-Spread* remains similar to the RDR, RRR, RDN, and RRN policies that spread the jobs across the machine.

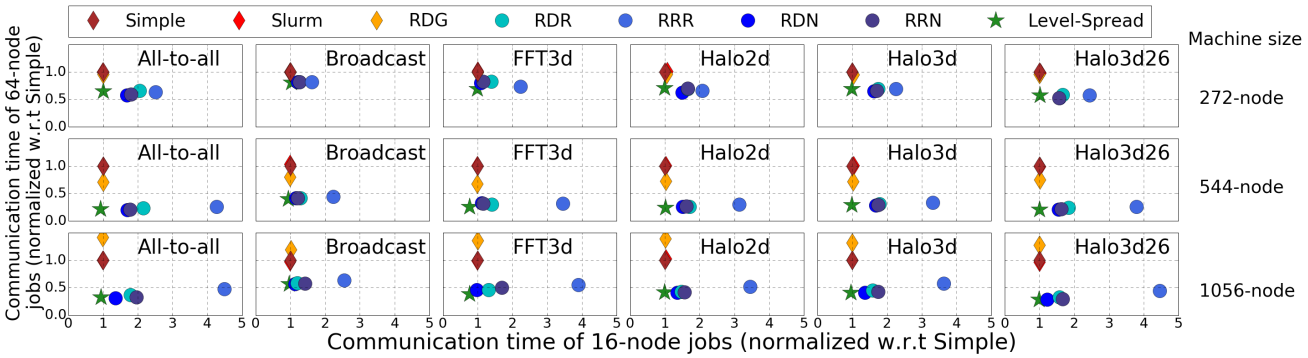
On the other hand, the diamonds in Fig. 7 lie in the top-left part of each subfigure, showing that these grouping-strategy policies do not work well for the large jobs. Circles lie in the bottom-right part of each subfigure, showing that these spreading-strategy policies do not work well for the small jobs.

Comparing the columns in Fig. 7, we observe the consistency of the benefits of our *Level-Spread* policy across different communication patterns. Different rows in Fig. 7(a) illustrate the impact of global link bandwidth on performance. Increasing the global link bandwidth reduces the difference among the policies but does not change their ranking.

Different rows in Fig. 7(b) explore the impact of machine size on performance. *Level-Spread* consistently outperforms the others in all three machine sizes and all communication



(a) Each row uses a different global-to-local link bandwidth ratio in a machine with 17 groups and 16 nodes per group. Simple, Slurm, and RDG overlap with each other. In the first row, RRR's X-axis position is beyond 5, thus not in the figure.



(b) Each row uses a different machine size. The global-to-local link bandwidth ratio is 1.

Fig. 7: Communication time of heterogeneous workloads each composed of n small jobs (16-node) and n large jobs (64-node) with a message size of 1KB. The number n is determined by the target utilization level of 90%. In each subfigure, a point represents the results from running the heterogeneous workload using a specific allocation. The X-axis of the point represents the average communication time of the small jobs, and the Y-axis represents that of the large jobs. The star corresponds to the *Level-Spread* allocation policy; the diamonds correspond to the grouping-strategy policies; the circles correspond to spreading-strategy policies. Values are normalized with respect to the Simple allocation policy in each subfigure.

patterns. Increasing the machine size magnifies the difference among allocations. In the case of a 1056-node machine (33 groups), *Level-Spread* policy reduces the communication time of the 16-node jobs by 32% (Broadcast) to 64% (All-to-all).

In these simulations, changing machine utilization level from 90% to 70% has negligible influence on the relative performance of the eight allocation policies, so we omit the results for 70% utilization.

C. Impact of Communication Intensity

To explore the influence of communication intensity on the performance of different allocation policies, for all our workloads and parameter sets listed, we run simulations using 1KB and 100KB job message sizes. We find that as long as the communication intensity is homogeneous for all jobs in the workload, the impact of job message size on the relative performance of different allocation policies is negligible. Simultaneously increasing message size of all jobs from 1KB to 100KB only increases the communication time of every job by approximately 100 folds. Due to the similarity of these results to Fig. 4 and Fig. 7, we do not depict these results.

In Fig. 8, we explore the cases where the communication intensity of 16-node jobs is different from the intensity of 64-node jobs. We experiment on a 272-node machine and simulate different combinations of message sizes.

From top to bottom in Fig. 8, we gradually increase the ratio of communication intensity between the 16-node jobs and the 64-node jobs. These plots demonstrate a clear trend that increasing the communication intensity of the large (64-node) jobs worsens the performance of RDR, RRR, RDN, and RRN policies on the small (16-node) jobs. This is because when the tasks of the small jobs are spread into many groups, the intensive communication among the tasks of the large jobs drastically delays the communication of the small jobs. Conversely, increasing the communication intensity of the small jobs significantly worsens the performance of RDR, RRR, RDN, and RRN policies on the large jobs. This is because in these spreading policies, the communication among the tasks of the large jobs are delayed by the communication-heavy small jobs. However, for Simple, Slurm, RDG, and *Level-Spread* policies, the small jobs are less affected by the interference with the large jobs because the small jobs by these policies do not stress global links. These results agree with the findings of Yang *et al.* [9].

The results in Fig. 8 demonstrate the effectiveness of *Level-Spread* in various communication intensities. We have also

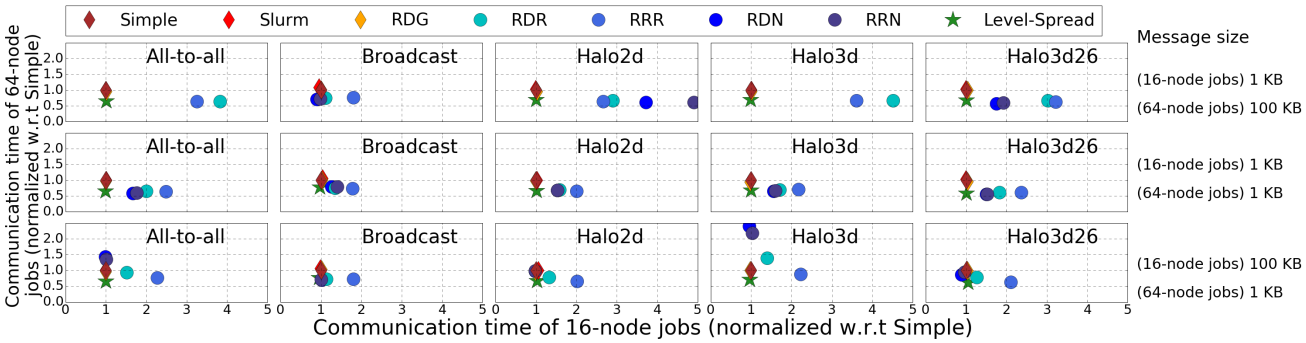


Fig. 8: Varying the communication intensity in terms of message size of the jobs. Here, we simulate a heterogeneous workload composed of three 16-node jobs and three 64-node jobs on a 272-node machine. We repeat the simulations with six communication patterns and different message sizes. From top to bottom, we gradually increase the ratio of communication intensity between the 16-node jobs and the 64-node jobs. In the first row, some circles are beyond the range of the X-axis and thus not displayed.

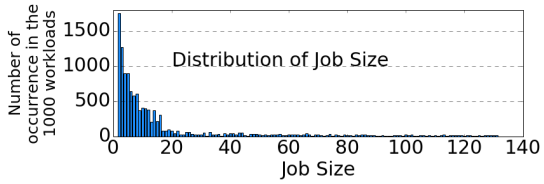


Fig. 9: Counted occurrence of job sizes in the 1000 randomly generated workloads used in Sec. VI-D.

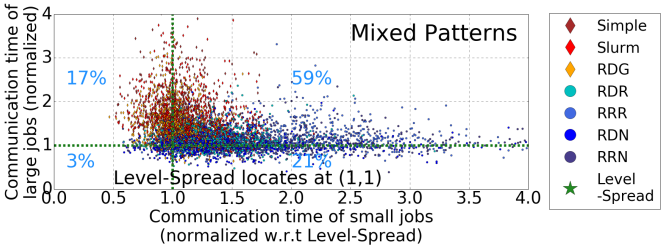


Fig. 10: Results from 1000 randomly generated workloads with mixed job sizes and communication patterns. Each point represents one allocation policy in one workload. Values are normalized with respect to *Level-Spread*.

verified that these conclusions on communication intensities are valid for different machine parameters listed in Table I.

D. Mixed Job Sizes and Communication Patterns

In Sec. VI-B, we have examined the performance of different allocation policies using workloads composed of 16-node jobs and 64-node jobs. To verify that our results can be generalized to other job sizes and mixed communication patterns, we use a broader set of 1000 randomly generated workloads, each composed of two types of jobs (small and large). The following simulations use a 272-node machine with 16 nodes per group (4 nodes per router and 4 routers per group) and a global-to-local links bandwidth ratio of 1. Machine utilization level is not fixed and depends on the generated workload.

To generate the workloads, we first randomly select an integer between 17 and 136 ($= 272/2$) as the size of the large jobs. We choose the upper limit as 136-node because an individual job is commonly restricted from occupying more than half of a machine. Next, a random integer between 2 and 16 is chosen as the size of small jobs. Then, the number of the small jobs and the number (which can be different) of the large jobs are selected randomly under the restriction of machine size. Fig. 9 shows the overall distribution of job sizes in these 1000 random workloads, which qualitatively matches the distribution of job sizes in real HPC systems (e.g., [36]). The communication pattern for each of the two types of jobs is randomly selected. In these simulations, we allocate small jobs before the large jobs in a given workload, and start running all jobs simultaneously. All jobs use a message size of 1KB.

In Fig. 10, the X-/Y-axis of each point represents the average communication time of the small/large jobs in one workload using one allocation policy. For each workload, the values are normalized with respect to the average communication time achieved using the *Level-Spread* policy. The dashed green lines split the graph into four parts, and a blue number shows the percentage of points in each part.

Fig. 10 shows that the grouping-strategy allocation policies (Simple, Slurm, and RDG) benefit the small jobs over the large jobs. Conversely, the spreading-strategy policies (RDR, RRR, RDN, RRN) benefit the large jobs at a cost of higher small-job communication overhead. *Level-Spread*, located at coordinate (1,1), combines the advantages of both grouping and spreading. Only in 3% of all cases, a baseline policy is strictly better than *Level-Spread*, i.e., for both the small and the large jobs. Meanwhile, in 59% of all cases, *Level-Spread* is strictly better than the baselines. In the best case, *Level-Spread* reduces communication time by 71%. Averaging over both small and large jobs and over all 1000 workloads, *Level-Spread* reduces communication time by 16%.

In addition, we run random-workload simulations where only a single communication pattern is used. For each communication pattern, we generate 100 workloads composed of small and large jobs. Our results show that for All-to-all, Halo3d, Halo3d26, Halo2d and Broadcast workloads the percentage of cases where a baseline policy is strictly better than *Level-Spread* is 1%, 1%, 3%, 4%, and 8%, respectively.

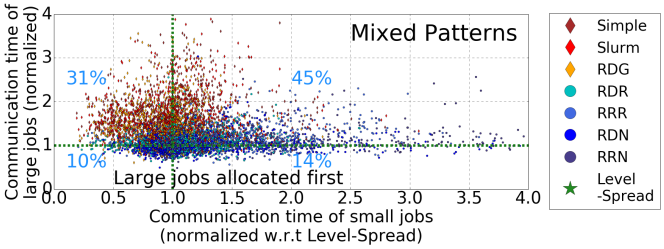


Fig. 11: In an adverse scheduling decision where the allocation of large jobs (jobs that cannot fit in a single group) are prioritized, *Level-Spread* at least performs as well as the baselines.

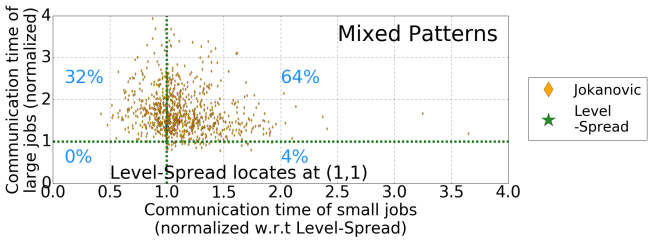


Fig. 12: Comparing *Level-Spread* policy with Jokanovic's policy using 1000 random workloads with mixed job sizes and communication patterns.

E. Influence of Scheduling on the Level-Spread Policy

In HPC systems, the order of allocating pending jobs is decided by a job scheduler, and this order affects where the jobs are allocated. For our *Level-Spread* policy, the scheduled order of pending jobs may influence the performance due to the job-size-awareness of *Level-Spread*. For example, assume a 15-node job and a 50-node job are pending on an empty 272-node dragonfly system whose nodes per group is 16. Using *Level-Spread*, if the 50-node job is scheduled first, it will be spread into different groups, occupying 2 or 3 nodes in each group. In this case, the 15-node job won't be able to fit in any group and will be also spread, converging to the RRN policy.

To evaluate the performance of *Level-Spread* in an adverse scheduling decision, we generate 1000 random workloads composed of two types of jobs similar to Sec. VI-D, but this time, we schedule the large jobs before the small jobs. To clarify, we allocate all jobs following the scheduled order and then start running them simultaneously. Fig. 11 shows that prioritizing large jobs in scheduling slightly moves all the points of baseline policies toward the left compared to Fig. 10, making the performance of *Level-Spread* closer to the four spreading-strategy allocations. This shows that with an adverse scheduling decision where large jobs are scheduled first, *Level-Spread* performs at least as well as the baselines.

F. Comparison with Jokanovic's Allocation Policy

In the previous sections, we compare *Level-Spread* with state-of-the-art policies for dragonfly networks. There are also allocation policies for other network topologies such as Jokanovic's policy [14] for fat-tree networks (see Sec. IV). To compare *Level-Spread* with Jokanovic's policy, we simulate 1000 random workloads similar to Sec. VI-D (parameters are kept the same). Fig. 12 shows that in 64% of the workloads,

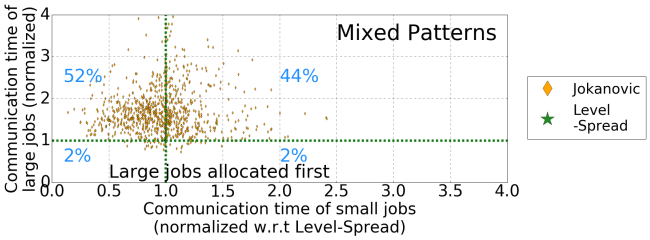


Fig. 13: Comparing *Level-Spread* policy with Jokanovic's policy using 1000 random workloads where large jobs are always allocated prior to small jobs.

Level-Spread performs strictly better than Jokanovic's policy (i.e., for both small jobs and large jobs). Jokanovic's policy does not perform strictly better than *Level-Spread* in any of these workloads. While Jokanovic's policy is good for small jobs, it is significantly worse than *Level-Spread* for large jobs.

The reason why Jokanovic's policy does not perform well in dragonfly networks is that dragonflies are not very sensitive to system fragmentation, owing to its low diameter and the all-to-all inter-group connections. As discussed in Sec. III-A, dragonfly networks benefit from a more balanced network traffic when we spread large jobs. Therefore, spreading large jobs, as done by *Level-Spread*, gives better performance than Jokanovic's policy, which groups large jobs together.

Similar to Sec. VI-E, we also run simulations on 1000 random workloads where the large jobs are scheduled prior to small jobs. Fig. 13 demonstrates that even with this adverse scheduling decision for *Level-Spread*, our *Level-Spread* policy continues to outperform Jokanovic's policy.

VII. Conclusions and Future Work

On dragonfly networks, compactly allocating the tasks of a parallel application to harness locality and spreading the tasks to balance congestion on network links are two allocation strategies to reduce communication latency. Existing allocation policies resemble only one of these two strategies. To combine the benefits of these two strategies, we propose *Level-Spread allocation policy*, which finds the lowest network level (router, group, or machine) a job can fit in and spreads the job throughout that level. Our allocation policy combines the advantages of existing policies for dragonfly networks.

To evaluate *Level-Spread*, we conduct extensive simulations with a broad range of workloads. We compare *Level-Spread* with eight other allocation policies, and conclude that *Level-Spread* outperforms the state-of-the-art by 16% on average (and up to 71%) in terms of communication time. To examine the applicability of our policy under different conditions, we conduct simulations with various dragonfly configurations, global link bandwidths, job communication intensities, and communication patterns. Our results validate the generality of *Level-Spread*, and we show that the performance gain from our policy further increases when the machine size increases or the global link bandwidth decreases.

Since our allocation policy selects nodes based on job size, the order of scheduling for a set of jobs may affect the

results. Using simulations, we demonstrate that scheduling the smaller jobs before the larger ones is beneficial for *Level-Spread*. In future work, a joint optimization of allocation and scheduling can further improve the performance of *Level-Spread* as well as any other potential size-aware allocation policies on hierarchical network topologies.

ACKNOWLEDGMENT

This work has been partially funded by the EU (FEDER) and the Spanish MINECO, under grant TIN 2015-65277-R. This work has also been partially funded by Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under Contract DE-NA0003525.

REFERENCES

[1] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs, "There goes the neighborhood: performance degradation due to nearby jobs," *Proceedings of the International Conf. on High Performance Computing, Networking, Storage and Analysis, (SC)*, pp. 41:1—-41:12, 2013.

[2] V. J. Leung, E. M. Arkin, M. A. Bender, D. Bunde, J. Johnston, A. Lal, J. S. B. Mitchell, C. Phillips, and S. S. Seiden, "Processor allocation on Cplant: Achieving general processor locality using one-dimensional allocation strategies," *IEEE International Conf. on Cluster Computing, (CLUSTER)*, pp. 296–304, 2002.

[3] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," *International Symposium on Computer Architecture (ISCA)*, pp. 77–88, 2008.

[4] M. Belka, M. Doubet, S. Meyers, R. Momoh, D. Rincon-Cruz, and D. P. Bunde, "New link arrangements for dragonfly networks," *IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, pp. 17–24, 2017.

[5] P. Fuentes, E. Vallejo, C. Camarero, R. Beivide, and M. Valero, "Network unfairness in dragonfly topologies," *Journal of Supercomputing*, vol. 72, no. 12, pp. 4468–4496, 2016.

[6] P. Faizian, S. Rahman, A. Mollah, X. Yuan, S. Pakin, and M. Lang, "Traffic pattern-based adaptive routing for intra-group communication in dragonfly networks," *IEEE Annual Symposium on High-Performance Interconnects (HOTI)*, 2016.

[7] N. Jain, A. Bhatele, S. White, T. Gamblin, and L. V. Kale, "Evaluating HPC networks via simulation of parallel workloads," *SC*, pp. 154–165, 2016.

[8] T. Groves, R. E. Grant, S. Hemmer, S. Hammond, M. Levenhagen, and D. C. Arnold, "(SAI) Stalled, Active and Idle: characterizing power and performance of large-scale dragonfly networks," *CLUSTER*, pp. 50–59, 2016.

[9] X. Yang, J. Jenkins, M. Mubarak, R. B. Ross, and Z. Lan, "Watch out for the bully! Job interference study on dragonfly network," *SC*, pp. 750–760, 2016.

[10] F. Kaplan, O. Tuncer, V. J. Leung, S. K. Hemmert, and A. K. Coskun, "Unveiling the Interplay between Global Link Arrangements and Network Management Algorithms on Dragonfly Networks," *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 325–334, 2017.

[11] 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," *SC*, 2012.

[12] V. T. Chakaravarthy, M. Kedia, Y. Sabharwal, N. P. Kumar Katta, R. Rajamony, and A. Ramanan, "Mapping strategies for the PERCS architecture," *International Conf. on High Performance Computing (HiPC)*, 2012.

[13] N. Jain, A. Bhatele, X. Ni, N. J. Wright, and L. V. Kale, "Maximizing throughput on a dragonfly network," *SC*, pp. 336–347, 2014.

[14] A. Jokanovic, J. C. Sancho, G. Rodriguez, A. Lucero, C. Minkenberg, and J. Labarta, "Quiet neighborhoods: key to protect job performance predictability," *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 449–459, 2015.

[15] M. Kambadur, T. Moseley, R. Hank, and M. A. Kim, "Measuring interference between live datacenter applications," *SC*, p. 1161, 2012.

[16] R. Budiardja, L. Crosby, and H. You, "Effect of rank placement on Cray XC30 communication cost," *The Cray User Group Meeting*, 2013.

[17] E. Hastings, D. Rincon-Cruz, M. Spehlmann, S. Meyers, A. Xu, D. P. Bunde, and V. J. Leung, "Comparing global link arrangements for dragonfly networks," *CLUSTER*, pp. 361–370, 2015.

[18] M. Garcia, E. Vallejo, R. Beivide, M. Odriozola, and M. Valero, "Efficient routing mechanisms for dragonfly networks," *Proceedings of the International Conf. on Parallel Processing*, pp. 582–592, 2013.

[19] N. Jiang, J. Kim, and W. J. Dally, "Indirect adaptive routing on large scale interconnection networks," *ACM SIGARCH Computer Architecture News*, vol. 37, p. 220, 2009.

[20] B. Prisacari, G. Rodriguez, P. Heidelberger, D. Chen, C. Minkenberg, and T. Hoefler, "Efficient task placement and routing in dragonfly networks," *ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, pp. 129–140, 2014.

[21] O. Tuncer, Y. Zhang, V. J. Leung, and A. K. Coskun, "Task Mapping on a Dragonfly Supercomputer," *IEEE High Performance Extreme Computing Conf. (HPEC)*, 2017.

[22] B. Prisacari, G. Rodriguez, C. Minkenberg, M. Garcia, E. Vallejo, and R. Beivide, "Performance optimization of load imbalanced workloads in large scale dragonfly systems," *IEEE International Conf. on High Performance Switching and Routing (HPSR)*, 2016.

[23] Slurm, "Slurm's job allocation policy for dragonfly network," 2016, [Line 1634-1932; accessed 30-March-2017]. [Online]. Available: https://github.com/SchedMD/slurm/blob/master/src/plugins/select/linear/select_linear.c

[24] A. Bhatele, N. Jain, Y. Livnat, V. Pascucci, and P. T. Bremer, "Analyzing network health and congestion in dragonfly-based supercomputers," *IPDPS*, pp. 93–102, 2016.

[25] M. Mubarak, C. D. Carothers, R. Ross, and P. Carns, "Modeling a million-node dragonfly network using massively parallel discrete-event simulation," *SC Companion: High Performance Computing, Networking Storage and Analysis (SCC)*, pp. 366–376, 2012.

[26] B. Prisacari, M. Garcia, E. Vallejo, and R. Beivide, "Performance implications of remote-only load balancing under adversarial traffic in Dragonflies," *International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip*, 2014.

[27] A. F. Rodrigues, E. CooperBalls, B. Jacob, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, and P. Rosenfeld, "The structural simulation toolkit," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, p. 37, 2011.

[28] J. Wilke, J. Bennett, H. Kolla, K. Teranishi, N. Slattengren, and J. Floren, "Extreme-Scale viability of collective communication for resilient task scheduling and work stealing," *IEEE/IFIP International Conf. on Dependable Systems and Networks (DSN)*, pp. 756–761, 2014.

[29] M.-Y. Hsieh, A. Rodrigues, R. Riesen, K. Thompson, and W. Song, "A framework for architecture-level power, area, and thermal simulation and its application to network-on-chip design exploration," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, p. 63, 2011.

[30] K. D. Underwood, M. Levenhagen, and A. Rodrigues, "Simulating red storm: challenges and successes in building a system simulation," *IPDPS*, pp. 1–10, 2007.

[31] E. S. Hertel, R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. McGlaun, S. V. Petney, S. A. Silling, P. A. Taylor, and L. Yarrington, *CTH: A software family for multi-dimensional shock physics analysis*. Springer Berlin Heidelberg, 1995, pp. 377–382.

[32] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of computational physics*, vol. 117, no. 1, pp. 1–19, 1995.

[33] K. Antypas, J. Shalf, and H. Wasserman, "NERSC 6 workload analysis and benchmark selection process," *Lawrence Berkeley National Laboratory*, pp. 1–45, 2008.

[34] S. Sreepathi, E. D'Azevedo, B. Philip, and P. Worley, "Communication Characterization and Optimization of Applications Using Topology-Aware Task Mapping on Large Supercomputers," *ACM/SPEC International Conf. on Performance Engineering (ICPE)*, pp. 225–236, 2016.

[35] K. Asanovic, B. C. Catanzaro, D. A. Patterson, and K. A. Yelick, "The Landscape of Parallel Computing Research : A View from Berkeley," *EECS Department University of California Berkeley Tech Rep UCBEECS2006183*, vol. 18, p. 19, 2006.

[36] D. G. Feitelson, D. Tsafrir, and D. Krakov, "Experience with using the Parallel Workloads Archive," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2967–2982, 2014.