# An Analysis of Long-tailed Network Latency Distribution and Background Traffic on Dragonfly+

Majid Salimi Beni and Biagio Cosenza

Department of Computer Science, University of Salerno
Salerno, Italy.
{msalimibeni,bcosenza}@unisa.it

**Abstract.** Modern computing systems are highly affected by large performance variability, resulting in a long tail in the distribution of the network latency. For communication-intensive applications, the variability comes from several factors such as the communication pattern, job placement strategies, routing algorithms, and most importantly, the network background traffic. Although recent high-performance interconnects such as Dragonfly+ try to mitigate this variability by employing advanced techniques such as adaptive routing or topological improvements, the long tail is still there.
This paper analyzes the sources of performance variability on a large-scale computing system with a Dragonfly+ network. Our quantitative study investigates the impact of several sources, including the locality of job placement, the communication pattern, the message size, and the network background traffic. To tackle the difficulty in measuring the network background traffic, we propose a novel heuristic that accurately estimates the network traffic and helps to identify those highly-varying communications that contribute to the long tail. We have experimentally validated our proposed background traffic heuristic on a collection of pattern-based microbenchmarks as well as two real-world applications, HACC and miniAMR. Results show that the heuristic can successfully predict most of those runs in long-tail at job submission time on both microbenchmarks and real-world applications.

**Keywords:** MPI · Interconnect · Congestion · Dragonfly+ · Topology.

## 1 Introduction

The growing gap between communication and computation in high-performance computing emphasizes the importance of optimized data communication. It is today well-understood that, to reach the Exascale, computing systems should provide high-performance network interconnects that deliver both high bandwidth and low latency.

The Dragonfly+ topology [47] is a modern hierarchical interconnect that has been recently introduced as an extended implementation of Dragonfly [30]. Such interconnect not only provides better network utilization and scalability in comparison to Dragonfly but also improves router buffer utilization [47]. However, despite Dragonfly+'s improvements compared to its predecessor, it still suffers

from performance variability, especially with higher network congestion. Performance variability affects both system and applications' performance, and the batch scheduler must have a more precise estimation of applications' runtime to make accurate scheduling decisions [67, 53].

Several users use large-scale compute clusters simultaneously, with different utilization patterns regarding program workflow, number of nodes, and data communication. While single-node computes units are typically not shared between users, the network is a shared resource. Network elements such as routers and links, shared among several jobs, are subject to contention. They negatively impact users' program performance by degrading I/O and slowing communication time. To address these issues, recent work has focused on monitoring, predicting, and balancing network traffic [58, 12, 33, 32], as well as taking topological and network designing aspects into account [7, 52, 22, 9]. In fact, the network has been identified as the main reason for performance variability [48, 11, 5, 10].

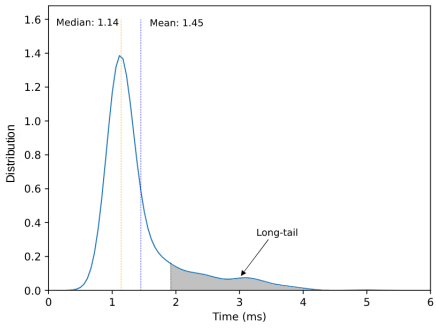



Fig. 1: Long-tail of the latency distribution on Dragonfly+.

### 1.1 Motivations

As performance variability is affected mainly by the network, it is essential to understand how network latency behaves on modern large-scale compute clusters. Figure 1 shows the frequency distribution of 1000 iterations of a latency test (MPI_Reduce in this case) on 16 nodes of the Marconi100 compute cluster with a Dragonfly+ topology. Interestingly, the results show a so-called *long-tailed* distribution. While a majority of the communication latencies are distributed around the median, more than 15% of the runs' latencies are larger than the 85th percentile (1.92 ms). The presence of such a long tail in the distribution also indicates that the distribution is not symmetric (e.g., not Gaussian), and there is a large gap between the mean and median. Also, the long tail negatively impacts the overall network performance by making the job execution highly unpredictable. While such performance variability is related to several network-related factors, our work aims to analyze the main reasons behind such performance degradation, from the application's communication patterns to the external network traffic involving all users.

At the topology level, our work focuses on the Dragonfly+, which has better network utilization [47] than Dragonfly (known to suffer from performance variability [50, 37]) and it is becoming a common topology in newly developed supercomputers [42, 34].

### 1.2 Contributions

This paper conducts a performance variability study on a large-scale compute cluster with Dragonfly+ topology. The study comprises the analysis of several known sources of performance variability, in particular network-related aspects, including: different communications patterns, the impact of message size, the locality of job placement, and the effect of network background traffic generated by other users. The latter, in particular, is difficult to measure; to this end, we propose an easy-to-measure heuristic that estimates such traffic. As a part of the study, we further point out the effect of the adaptive routing strategy on the communication performance of Dragonfly+.

To the best of our knowledge, this is the first work that analyzes Dragonfly+ performance variability on a real supercomputer. While most related work relies on simulating background traffic [28, 57], our approach is based on real-world data of background traffic extracted from a large-scale compute cluster. Insights from this analysis provide valuable feedback for job placement policy implementations on Dragonfly+ as well as network design for large-scale clusters.

The main contributions of this paper are:

- The first detailed **analysis of communication performance on a large-scale Dragonfly+ network based on real-world data**: We analyze different inter-node communication scenarios and show the performance variability of microbenchmarks with varying job placements.
- A **novel heuristics for background traffic estimation**, which is easy to measure and based on information known at job submission time.
- A comprehensive **correlation analysis between estimated background traffic and the communication performance**, with different communication patterns and message sizes.
- An **evaluation of the background traffic's impact on the long-tail of the latency distribution**.
- Further extension of the evaluation on **two communication-intensive real-world applications**: HACC [1] and miniAMR.

The rest of the paper is organized as follows: Section 2 and 3 introduce, respectively, related work and experimental setup. Section 4 presents our analysis of latency distribution, and Section 5 describes our background traffic measurement approach and its analysis. Section 6 is the discussion, and Section 7 concludes the paper.

## 2 Related Work

A large part of the execution time of HPC applications is spent on transferring data between nodes; for this reason, considerable research efforts have been paid

[1] Hardware Accelerated Cosmology Code

to investigating network topologies [39, 4, 26, 20] and, on the application side, studying, analyzing, and optimizing communication on top of existing topologies [51, 2, 54, 46, 16, 18, 55].

Performance variability is often correlated with heavy-tailed distributions, which are probability distributions whose tails are not exponentially bounded [3]. In fact, when scaling up and increasing the complexity of a computing system, the tail of the latency distribution, which is not long in small systems, becomes more dominant at the large scale [14].

Bhatele et al. [5] analyzed the performance variability of Dragonfly with periodic system profiling of mini-applications; based on this analysis, they trained a machine learning model that predicts future executions. Groves et al. [19] studied the performance variability of the MPI_Allreduce collective in the Aries Dragonfly network and considered the relationship between different metrics such as process count, Aries counters, and message size with communication time, and showed the impact of background traffic on the performance.

Research on performance variability has investigated locality aspects and studied how topological locality and communication patterns affect different applications' performance [63]. Other research, however, considered other metrics such as network designs [13, 60, 44], routing strategies [8, 50, 27, 40, 38, 15], congestion control [35, 45] and background traffic [65]. Wilke et al. [61] discuss and compare existing challenges of Dragonfly and Fat-tree and show how different configurations and routing algorithms may affect QoS. They further illustrate the performance variability of Dragonfly while having various background traffic and different routing strategies. Alzaid et al. [1] have explored the Dragonfly network and measured the impact of different link arrangements between nodes and routing strategies on communication between nodes. They showed how data transfer through different links might be affected while the links tolerate different bandwidths.

Job allocation strategies have been recognized as a determinant factor in communication performance [29, 36]. Level-Spread proposed by Zhang et al. [66] is a job allocation policy on Dragonfly that puts jobs in the minor network level that the current job can fit in to not only benefit from the node adjacency but also balance link congestion. Brown et al. [6] analyzed the relation between MPI communications and I/O traffic in Fat-tree networks; their analysis considers different parameters such as job allocation policies, message sizes, communication intervals, and job sizes. Wang et al. [59] have performed a comparative analysis of network interference on applications with nearest-neighbor communication patterns, considering various job placement strategies on Dragonfly. They show that having a trade-off between localized communication and a balanced network in job placement can reduce network interference and alleviate performance variability. In another work [58], they carried out an in-depth performance analysis on Dragonfly and demonstrated how balanced network traffic and localized communication could impact different workloads.

Although related work has studied performance variability in Dragonfly, to the best of our knowledge, none of them have deeply investigated this variability

in Dragonfly+. Moreover, we specifically show how background traffic affects different communication patterns, i.e., which collectives are more vulnerable to background traffic. Unlike most related work on background traffic, our analysis is based on real-world data (experiments have been conducted during a three-month time span at different times in order to have different background traffic) rather than simulations. Hence, the background traffic is generated by other users we have no control over, and we are not producing such traffic artificially.

## 3 Experimental Setup

Our analyses have been performed on a large-scale compute cluster, Marconi100 [34], available at the CINECA supercomputing center, which is currently ranked 18th in the TOP500 ranking [56].

### 3.1 Computing

The Marconi 100 cluster is an IBM Power System AC922 [43] consisting of 980 nodes, each of which is equipped with two IBM POWER9 AC922 multicore processors with 16 cores at 2.6 (3.1 turbo) GHz and four NVIDIA Volta V100 GPUs with 16GB, and 256 GB of per-node memory. All in all, the total number of CPU cores is 347,776, and it provides 347776 GB of memory.

### 3.2 Network

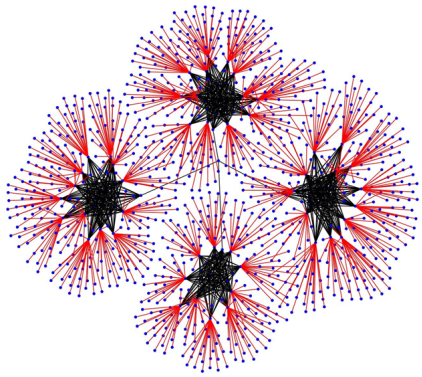


Fig. 2: The Dragonfly+ topology in Marconi100.

The internal interconnect of Marconi100 is a Mellanox InfiniBand EDR Dragonfly+. Figure 2 presents the Dragonfly+ topology implemented in this supercomputer. As shown, there are four large groups of nodes, each of which is called an *island*. Within islands, there are smaller groups of nodes connected to one switch called *groups*. The main topological difference between Dragonfly and Dragonfly+ is that in Dragonfly+, intra-island routers are connected as a bipartite graph to improve the scalability.

It is worth mentioning that the Operating System is Red Hat Enterprise 7.6, IBM Spectrum-MPI 10.4 [25] is installed on the cluster, and SLURM [62] has the duty of resource management on this system. In addition, Adaptive Routing [17] is the default routing strategy used to prevent contention of the links and handle failures on the hardware.

### 3.3 Microbenchmarks and Applications

The main analysis and evaluation are done based on the OSU collection of microbenchmarks [41], which consists of three collectives, to which we added two real-world applications as summarized in Table 1. Moreover, to show the performance variability, each experiment is repeated in 1-millisecond intervals 1000 times in a loop (as suggested by [24] to perform at least 300 iterations), and, in all experiments, 1 MPI process is assigned to each physical node to leave other cores for the OS. Also, 16 physical nodes are allocated to the cluster in collective communications and application evaluations to partially involve all the islands in the communication.

| **Benchmark/App** | **Description** | **Evaluated sizes** |
|---|---|---|
| Broadcast | Program calling Spectrum MPI_Bcast | $2^2, 2^{10}, 2^{15}, 2^{20}$(bytes) |
| Reduce | Program calling Spectrum MPI_Reduce | $2^2, 2^{10}, 2^{15}, 2^{20}$(bytes) |
| All-to-All | Representative of Spectrum MPI_Alltoall | $2^2, 2^{10}, 2^{15}, 2^{20}$(bytes) |
| HACC [21] | Includes various communication patterns | 10M particles |
| miniAMR [23] | Includes various communication patterns | 4K 3D blocks |

Table 1: Benchmarks and applications used for the analysis.

## 4 Network Latency Distribution Analysis

This section provides an analysis of the network latency on a Dragonfly+. First, we show the performance variability considering different locality levels for node allocation. Then, we show how the performance of microbenchmarks is affected when having different job allocation scenarios. Note that to make sure we are using the best-fitting distribution with minimum error in distribution plots, more than 100 different distributions have been fitted to the data.

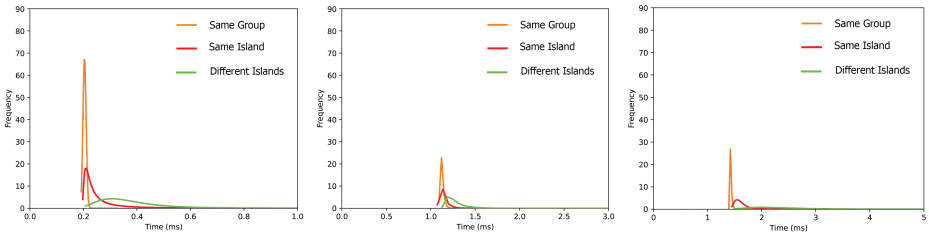
### 4.1 Job Placement Locality and Performance Variability

Performance variability is the difference in an individual program's performance in consecutive executions. This section shows the impact of different job placement (node allocation) strategies on performance variability.

In our analysis, we consider three locality levels according to the Dragonfly+ topology and analyze the performance variability when having the following three node allocation scenarios:

a) **Same Group**: In this case, all required nodes are allocated in a single group. Therefore, only one network switch is involved in the communication between every two nodes.

b) **Same Island**: Nodes are allocated on one island, but they are distributed across different groups of that island. Hence, there is less locality than in the previous scenario.
c) **Different Islands**: Nodes are distributed on different islands. In this case, there is no limitation on allocating nodes; they are allocated everywhere on different islands and groups. In doing so, less locality is imposed.




(a) Broadcast (b) Reduce (c) AlltoAll

Fig. 3: Communication time frequency distribution of collective communications for 1000 iterations, with different allocation locality scenarios.

According to the defined locality levels, we focus on the role of both communication patterns and job placement on the performance variability and long-tail. In fact, we analyze different communication patterns to understand how they affect performance variability. The selected microbenchmarks include *one-to-all* (MPI Broadcast), *all-to-one* (MPI Reduce), and *all-to-all* (MPI AlltoAll).

We refined the analysis with a by-pattern study as shown in figure 3. This figure shows the frequency distribution of under-study collectives with different allocations on 16 nodes. For the *same group* job placement, all 16 nodes are allocated on the *same group* and connected through a single switch. For *different islands* mode, four nodes are allocated on each island in different groups. As illustrated, Broadcast (figure 3a) shows the best performance and shortest tail for all three allocation strategies; in fact, it benefits local communications more than other patterns, especially for the *same group*: it is not only faster than others (average time= 0.2), but also its peak is higher, which means that communication times of different iterations are very similar and there is a low performance variability. In figure 3a, the peaks of *different islands* and *same island* are 19 and 6, respectively, and they possess a peak much lower than the *same group* (68). However, they still show higher peaks than the correspondings in Reduce and AlltoAll. For the Reduce (figure 3b), the average communication times of the *same group* and *same island* are almost the same (1.17 and 1.18 ms, respectively). However, with *different islands* we observe a slower average communication time (1.4 ms) and a much longer tail, reaching 10 ms. Finally, AlltoAll (figure 3c) is the slowest and most variable collective when all the nodes

are on *different islands*. Its frequency distribution shows a very long tail (notice that the end of its tail is not shown in the figure), with a maximum observed communication time reaching 13 ms and a peak of 2.

Although allocating all nodes on the *same group* has been beneficial for collective communications, the number of nodes in each group of Dragonfly+ is limited (up to 20 nodes in Marconi 100), and the job scheduler cannot exclusively allocate to the *same group* more than the existing physical nodes. Even worse, large-scale compute clusters are typically used by several users that submit multiple jobs; in fact, very often, other nodes in the same group are already allocated by other users' jobs. In such cases, the job scheduler should necessarily allocate a job to nodes on different groups of that island or other islands unless we are willing to wait hours or even days until all the nodes in the same group are idle.

By default, SLURM [49] tries to place jobs on the currently idle nodes in the same group if the user does not specify particular nodes (in the host file). Because of the limited amount of idle nodes that can be found in the same group, SLURM's job scheduler looks for the switches (groups) with the fewest number of idle nodes and chooses the idle nodes connected to that switch, and repeats this process until it assigns all the requested nodes. So, based on the requested number of nodes by the user and the availability of cluster nodes, it may decide to assign jobs to nodes on different groups of the same island, or it spans over different islands, which the latter is the more probable scenario according to our observations.

## 5 Background Traffic Analysis

In real-world supercomputers, a single user does not operate on a dedicated system; instead, it submits jobs concurrently with other users. While resources such as computing nodes are typically allocated so that they are not shared between users at the same time, unfortunately, there is a resource for which some degree of contention is unavoidable: the network.

Intuitively, the larger the number of active jobs, the more probable the network congestion. More precisely, network congestion is more probable when users' jobs involve a larger number of nodes.

This section analyzes how the background traffic generated by other users' jobs affects the performance variability. In particular, we first define a simple heuristic that approximates the amount of network activity generated by other users' jobs. Successively, the analysis focuses on the correlation between background traffic with several communication patterns and message sizes.

### 5.1 Background Traffic Heuristic

The network congestion due to other users' activity is an essential cause of high-latency runs when using a large-scale compute cluster. We indicate with network background traffic: the external network traffic made by other users who are running their job simultaneously. To quantify how much such network activities impact the latency of our program communications, we have monitored

the SLURM job queue before executing our jobs (i.e., we queried the *squeue* command before program execution).

In this way, we obtained information regarding the number of running and pending jobs, running jobs' runtime, as well as the number of nodes allocated by each job. Since we have no information about pending jobs and it is unclear when they will be running, they are not considered in our background traffic analysis. Besides, the running jobs that allocate only one node are excluded from our calculations because they have no communication with other nodes and, therefore, no effect on the background traffic (we experimentally observed many jobs that only allocate one node). Therefore, only jobs with the running status that allocate at least two nodes have been taken into account.

To better understand the background traffic with a simple and countable metric, we define a simple heuristic named *background network utilization* ($b$), which is defined as the number of unique nodes allocated by the running jobs and whose allocation includes at least two nodes over all the available nodes of the cluster. In other words, it shows the ratio of nodes contributing to communication to all the physical cluster nodes.

Formally, the background network utilization $b$ ratio is defined as follows:

$$b = \frac{N_c}{N_t} \tag{1}$$

where:

$N_c$ : number of unique nodes contributing to communication
$N_t$ : total number of cluster physical nodes

In some cases, one node may be shared among different jobs by the scheduler in order to fully utilize its resources, e.g., each job takes a computation resource; which means that the node is being utilized by more than one communicating job, and we cannot count this node in our heuristic only once since the node produces higher background traffic. In order to take such cases into account, we count the shared node as many times it appears in the jobs' node lists that allocate more than two nodes. Hence, considering the appearance of some nodes more than once in the nodes list, the number of all running nodes can become larger than the cluster's physical nodes ($N_t$), which is a constant number. In an effort to resolve the problem and refine the heuristic, we consider the overhead of shared nodes by multiplying $b$ by a new ratio which is: the number of nodes contributing to communication (consider some nodes might be counted more than once) to all the allocated running nodes (Similarly, we count each node as many times they appear in the jobs' nodes list). By doing so, we ensure that we consider nodes contributing to different jobs and having communication with other nodes. Therefore, the refined version of the *background network utilization*, which will be considered in the rest of the paper, is defined as follows:

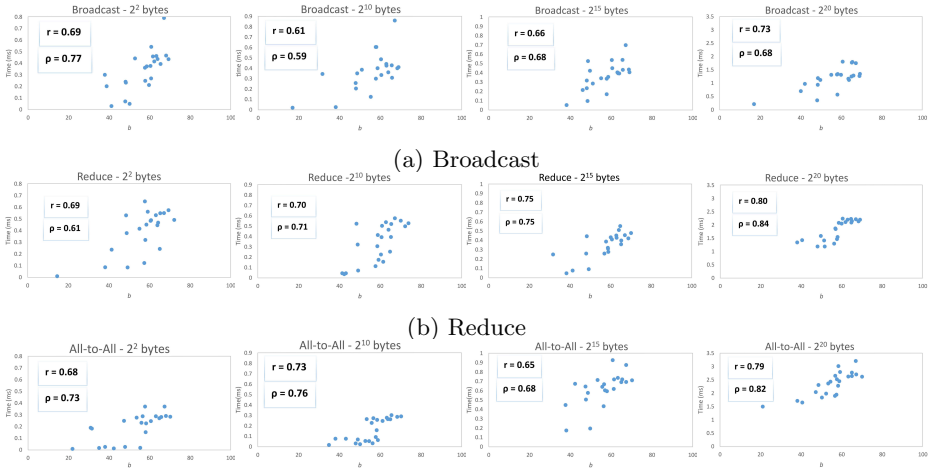
$$b = \frac{N_c}{N_t} * \frac{N'_c}{N_a} \tag{2}$$

where:

$N'_c$ : the number of nodes contributing to communication (containing duplication)
$N_a$ : all allocated running nodes (containing duplication)

Ideally, the value of $b$ is 1 (or 100, if the percentage is taken into account) if running jobs allocate all the nodes and all of them are actively involved in communication, while $b$ is 0 if non of the nodes are communicating or there is no active job at that moment. In order to make sure the measured $b$ is showing a more accurate background network utilization and it has not changed during the microbenchmark's execution, we perform the *squeue* query also after the execution of each test and capture the $b$ value only if the difference between two $b$ values calculated is less than a threshold (5% in our experiments).

Note that some other network-related metrics, such as vendor-provided counters, can be also measured in some clusters to make precise network congestion measurement. However, not in all compute clusters are these counters available or accessible by non-admin users. Moreover, using such counters, the proposed method would not be portable to other clusters with different network infrastructure vendors. Therefore, we rely on data provided by SLURM, which is available on most clusters.




(a) Broadcast

(b) Reduce

(c) AlltoAll

Fig. 4: The relation between background traffic ($b$) and the average communication time of different collectives with different message sizes.

### 5.2 Correlation Analysis

To evaluate how much the communication time is affected by the background traffic, we analyzed the correlation between the previously introduced $b$ metric and the communication time over many runs with different workloads in terms of data sizes and communication patterns. In the evaluation, we used the Pearson

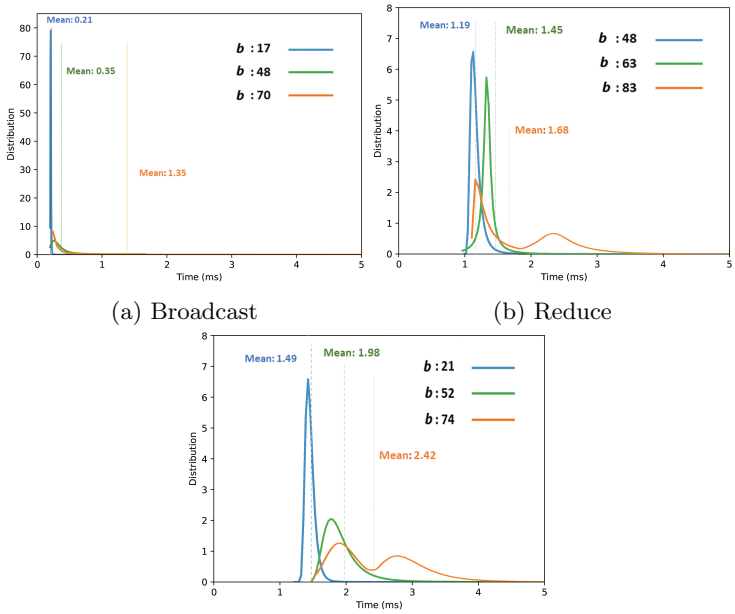
Correlation Coefficient ($r$) [31] and Spearman Rank Correlation ($\rho$) [64] to analyze the relation between the two metrics. While Pearson's correlation shows if there is a linear relationship between data, Spearman's correlation evaluates the monotonic relationships in the data. In both, $r, \rho$: $r = +1$ or $\rho = +1$ means that there is a strong positive correlation between the variables, while $r = 0$ or $\rho = 0$ means independent variables. Figure 4 shows the correlation between background network utilization $b$ and communication time for Broadcast (4a), Reduce (4b), and All-to-All (4c) pattern, with different data sizes on 16 nodes allocated on *different islands*. We do not explore point-to-point communication here since it is not significantly affected by the background traffic. There are 22 points on each plot, and each point represents the average time of 1000 iterations. Experiments are performed in a three months time frame and represent experiments under different cluster utilization, i.e., different recorded background network utilization.

As shown in Figure 4, the message transmission time is correlated with the *background network utilization* metric ($b$) and, overall, with increasing traffic, the communication time increases. In addition, as a general trend, with growing message size from $2^2$, $2^{10}$, and $2^{15}$ to $2^{20}$ bytes, the correlation between *background network utilization* and communication time becomes stronger, which means: the larger the data size is, the more the collective communication is affected by background traffic. Further, the correlations in Reduce collective for larger data ($2^{15}$ and $2^{20}$ bytes) are higher than in others, meaning that in this collective, the communication time is highly dependent on the background traffic. Also, comparing the Pearson and Spearman correlations, Spearman shows a better fit for our use cases since it usually shows a more strong correlation.

It is worth mentioning that although background traffic is an essential factor that affects performance variability in communication-intensive jobs running on supercomputers, it is not the only player. Other reasons come from MPI itself, system activities, background daemons, garbage collection, queuing activities in intermediate servers and network switches, etc. [48, 14]. Having said that, our *background network utilization* ratio is also an estimation relying on the obtainable information from other users. Hence, there might be possible errors in the measured runtimes, which is why some communications with smaller *background network utilization* have larger communication times, and the correlations are not +1.0 in figure 4.

### 5.3 The impact of background traffic on long-tail

We have seen how performance variability is affected by the network background traffic for specific input sizes and communication patterns. In this section, we go back to the motivation example and focus our analysis on the background traffic contribution to the long-tail effect. Figure 5 shows the frequency distribution of the execution time of 1000 iterations of 3 collectives on 16 nodes with message size $2^{20}$ bytes, with nodes allocated on *different islands*. For all three collectives, the higher the background network utilization, the lower the peak, and the longer the tail. For the Broadcast (5a) and $b = 0.17$ (17%), the peak is very high, and there is a significant gap in the distribution of the higher and lower traffics; with




(a) Broadcast (b) Reduce

(c) All-to-All

Fig. 5: Frequency distribution of communication times of 1000 iterations of Broadcast, Reduce, and All-to-All with different background network utilization.

higher background network utilization ($b = 0.70$), the tail of its corresponding distribution line is so long, which indicates that the communication performance is highly variable, ranging from 0.2 ms to 8 ms. Moreover, our experimental result reveals that the average execution time of 1000 iterations of Broadcast for $b = 0.70$ can be up to $6.4x$ larger than $b = 0.17$. Therefore, the Broadcast is highly affected by the background traffic, and, even if all the nodes are distributed on *different islands*, lower background traffic's performance can be as good as allocating all the nodes on the *same island*.

Similarly, in figures 5b and 5c, we observe that the distribution spreads at larger intervals with increasing background network utilization, and the tail becomes longer. For AlltoAll, especially when there is high background network utilization, the tail of the distribution is longer, the peak is lower, and the average communication time is larger than Broadcast and Reduce. Also, the mean of distribution with $b = 0.74$ is around $1.6x$ larger than $b = 0.21$. In addition, unlike others, in AlltoAll, a significant shift in the peak of the charts (Median) of different background network utilizations is observed. In fact, this shift in the peak of different traffics is because of the All-to-All's inherent communication intensity: in this pattern, all nodes send their data to the others, and more data is sent through the network, making the network links more congested.

Besides, for higher background network utilization of Reduce and AlltoAll, the frequency distribution becomes dual (bimodal), which means that the higher amounts of iterations mainly happen at two different times instead of one. This behavior is related to the adaptive routing algorithm employed in this Dragonfly+ network. In adaptive routing, the router has multiple paths to choose from for each packet. In this way, some packets traverse on the shortest (minimal) path, and some go through an alternative, longer (non-minimal) one. Hence, some communications happen slower than the majority due to the penalty of selecting the non-minimal path. As demonstrated in figures 5b and 5c, when the network tolerates higher background network utilization, going through the non-minimal path becomes more probable that this either causes the distribution tail longer or makes it dual. Note that we cannot change the routing strategy since we are performing our experiments on a real compute cluster. Overall, it is clear how the background traffic pushes the tail. While the adaptive routing strategy helps mitigate the problem, there are cases where the problem still exists, particularly when there is very high background traffic.

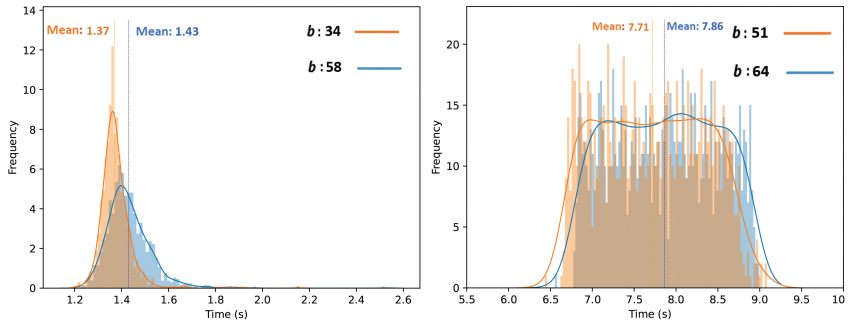
### 5.4 Application Analysis

So far, we have shown the impact of network background traffic and routing strategy on micro-benchmarks. In this section, we investigate the impact of background network utilization on two communication-intensive real-world applications that have shown to be affected by network congestion:

- HACC: a cosmology framework that performs n-body simulation to simulate the formation of structure in an expanding space.
- miniAMR: a mini-application that performs a stencil calculation on a unit cube computational domain.

Figure 6 shows the network latency distribution for HACC and miniAMR with both histogram and the frequency distribution. As shown in figure 6a for HACC, the average execution time and the peaks of $b = 34$ (the orange distribution) are 1.37 and 8.9, respectively. In contrast, for $b = 58$ (the blue distribution), the average time and peak reach 1.43 and 5.2, respectively. In other words, with a 24 percent increase in $b$, the average execution time increases by 4.4 percent. Moreover, both distributions in Figure 6a are single and bell-shaped. However, the blue line is broadly distributed, and its tail reaches 2.5, while the orange line's tail is 2.1.

On the other hand, in Figure 6b, when $b$ changes from 51% to 64% and changes by 13, the average goes from 7.71 to 7.86 (2% increase). In contrast to all the observations, in this figure, both plots have multiple peaks, and a different behavior has been observed. Regarding the previous analysis on the two applications [65], in HACC, around 67% of the overall execution time of the application belongs to MPI operations. However, a tiny fraction (0.1%) is related to blocking collective communications. On the contrary, in miniAMR, 27% of total time belongs to MPI operations, in which 9.2% of the overall execution time belongs to only MPI_Allreduce, which means miniAMR performs more collective communications with the All-to-All pattern.




(a) HACC (b) miniAMR

Fig. 6: Frequency distribution of 1000 iterations of HACC and miniAMR applications with two different background network utilization

As we have demonstrated in Figures 3 and 5, the All-to-All pattern is more prone to be affected by the network background traffic, and it has shown the flattest distribution when it is exposed to higher network background traffic in comparison to others. Moreover, the routing's effect can make its distribution bimodal. Looking over miniAMR's code, there are more than 10000 MPI_Allreduce operations which make the All-to-All pattern dominant. In Figure 6b, the distribution becomes flat-topped that the main reason is because of its dominant All-to-All pattern, and its distribution is an aggregation of all of its dominant MPI_Allreduce communication latencies. Having said that, the routing algorithm will also play a role here because of the communication intensity of the All-to-All pattern, and we could expect a multi-modal distribution because of mixing many MPI_Allreduce distribution patterns.

## 6 Discussion

Our analysis of network latency distribution on a large-scale compute cluster with Dragonfly+ topology led to several insights. In terms of node allocation, there is a remarkable discrepancy between the *same group* and the two other allocation policies. When all the nodes are allocated to a single group, there is only one hop between every two nodes, which makes the communication minimally affected by the global background traffic. For the same reasons, in this case, the minimal and non-minimal paths are the same for the adaptive routing (in contrast with the two other cases). So, it exhibits a latency distribution with the shortest tail and the higher peak. Hence, if there are enough available idle nodes on the *same group*, it is worth allocating all the required nodes there.

When analyzing the latency distribution according to the communication patterns, the Broadcast is the pattern that has significant benefit from the locality of the job allocation; in fact, results show that Broadcast has the shortest tail and higher peak and is faster than Reduce and All-to-All for both *same group* and *same island* allocations. However, when nodes are allocated on *different islands*, Broadcast is highly affected by the background traffic, showing a

very long tail compared to the cases with lower background traffic. Moreover, when the background traffic is very low, Broadcast's allocation performance on *different islands* can be as variable as allocation on the *same group*. Nevertheless, since the introduced background network utilization has been between 0.40 and 0.70 most of the time, there is very little chance of being in this situation. On the other hand, All-to-All is the pattern with the most extended tail when the job placement expresses little locality on Dragonfly+. Although its distribution when allocating on the *same group* is similar to the Reduce on the *same group*, when performing All-to-All on *different islands*, the distribution tail becomes very long due to the higher amount of communication in All-to-All.

Among all possible sources of performance variability, it has been shown that the background traffic is the key factor in the performance variability of different collectives on Dragonfly+. Usually, with the increase in background traffic, the communication time of collectives takes longer. Additionally, collective communication increases with higher background traffic and larger message sizes.

On top of that, we have experimentally observed a two-peak distribution of the communication latency typically due to the adaptive routing algorithm, which offloads some packets to an alternative, longer path under congestion. Finally, when analyzing the latency distribution of a real-world communication-intensive application, the distribution is mostly affected by its dominant communication pattern, and the overall average execution time increases with an increment in the network background traffic.

## 7 Conclusion

In this paper, we showed the performance variability of Dragonfly+ and analyzed the impact of background traffic on the long-tailed distribution for different communication patterns. We proposed a novel network background traffic estimation method that relies on the data gathered from the job scheduler's execution queue. We further showed the relation between performance variability and message size and demonstrated how the adaptive routing algorithm impacts the distribution.Overall, this study considers different metrics, including communication patterns, message sizes, job placement locality, and background traffic, to show how they contribute to performance variability and long-tail. We have experimentally validated our proposed background traffic heuristic on a large-scale cluster, a collection of pattern-based microbenchmarks, and two real-world applications.

The insights coming of this paper can help either the user or the scheduler to make more optimal decisions by first, estimating the network congestion according to the user-level information, and second, submitting the job at an appropriate time to have the minimum network interference.

## 8 Acknowledgments

This research has been partially funded by the European High-Performance Computing Joint Undertaking (JU) under grant agreement No. 956137 (LIGATE project).

## References


[1] Zaid Salamah A Alzaid, Saptarshi Bhowmik, Xin Yuan, and Michael Lang. "Global link arrangement for practical dragonfly". In: *Proceedings of the 34th ACM International Conference on Supercomputing*. 2020, pp. 1–11.

[2] Samar A Aseeri, Anando Gopal Chatterjee, Mahendra K Verma, and David E Keyes. "A scheduling policy to save 10% of communication time in parallel fast Fourier transform". In: *Concurrency and Computation: Practice and Experience* (2021), e6508.

[3] Majid Salimi Beni and Biagio Cosenza. "An Analysis of Performance Variability on Dragonfly+topology". In: *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. 2022, pp. 500–501. DOI: 10.1109/CLUSTER51413.2022.00061.

[4] Maciej Besta, Marcel Schneider, Marek Konieczny, Karolina Cynk, Erik Henriksson, Salvatore Di Girolamo, Ankit Singla, and Torsten Hoefler. "Fatpaths: Routing in supercomputers and data centers when shortest paths fall short". In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2020, pp. 1–18.

[5] Abhinav Bhatele, Jayaraman J Thiagarajan, Taylor Groves, Rushil Anirudh, Staci A Smith, Brandon Cook, and David K Lowenthal. "The case of performance variability on dragonfly-based systems". In: *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2020, pp. 896–905.

[6] Kevin A Brown, Nikhil Jain, Satoshi Matsuoka, Martin Schulz, and Abhinav Bhatele. "Interference between I/O and MPI traffic on fat-tree networks". In: *Proceedings of the 47th International Conference on Parallel Processing*. 2018, pp. 1–10.

[7] Kevin A Brown, Neil McGlohon, Sudheer Chunduri, Eric Borch, Robert B Ross, Christopher D Carothers, and Kevin Harms. "A Tunable Implementation of Quality-of-Service Classes for HPC Networks". In: *International Conference on High Performance Computing*. Springer. 2021, pp. 137–156.

[8] Ram Sharan Chaulagain, Fatema Tabassum Liza, Sudheer Chunduri, Xin Yuan, and Michael Lang. "Achieving the Performance of Global Adaptive Routing using Local Information on Dragonfly through Deep Learning". In: *ACM/IEEE SC tech poster* ().

[9] Qixiang Cheng, Yishen Huang, Meisam Bahadori, Madeleine Glick, Sebastien Rumley, and Keren Bergman. "Advanced routing strategy with highly-efficient fabric-wide characterization for optical integrated switches". In: *2018 20th International Conference on Transparent Optical Networks (ICTON)*. IEEE. 2018, pp. 1–4.

[10] Dean Chester, Taylor Groves, Simon D Hammond, Timothy R Law, Steven A Wright, Richard Smedley-Stevenson, Suhaib A Fahmy, Gihan R Mudalige, and Stephen Jarvis. "StressBench: A Configurable Full System Network and I/O Benchmark Framework". In: *IEEE High Performance Extreme Computing Conference*. York. 2021.

[11] Sudheer Chunduri, Kevin Harms, Scott Parker, Vitali Morozov, Samuel Oshin, Naveen Cherukuri, and Kalyan Kumaran. "Run-to-run variability on Xeon Phi based Cray XC systems". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2017, pp. 1–13.

[12] Daniele De Sensi, Salvatore Di Girolamo, and Torsten Hoefler. "Mitigating network noise on dragonfly networks through application-aware routing". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2019, pp. 1–32.

[13] Daniele De Sensi, Salvatore Di Girolamo, Kim H McMahon, Duncan Roweth, and Torsten Hoefler. "An in-depth analysis of the slingshot interconnect". In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2020, pp. 1–14.

[14] Jeffrey Dean and Luiz Andre Barroso. "The Tail at Scale". In: *Communications of the ACM* 56 (2013), pp. 74–80. URL: `http://cacm.acm.org/magazines/2013/2/160173-the-tail-at-scale/fulltext`.

[15] Peyman Faizian, Juan Francisco Alfaro, Md Shafayat Rahman, Md Atiqul Mollah, Xin Yuan, Scott Pakin, and Michael Lang. "TPR: Traffic Pattern-Based Adaptive Routing for Dragonfly Networks". In: *IEEE Transactions on Multi-Scale Computing Systems* 4.4 (2018), pp. 931–943.

[16] Shane Farmer, Anthony Skjellum, Ryan E Grant, and Ron Brightwell. "MPI Performance Characterization on InfiniBand with Fine-Grain Multithreaded Communication". In: *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC)*. IEEE. 2016, pp. 1102–1106.

[17] Christopher J Glass and Lionel M Ni. "The turn model for adaptive routing". In: *ACM SIGARCH Computer Architecture News* 20.2 (1992), pp. 278–287.

[18] Ryan E Grant, Matthew GF Dosanjh, Michael J Levenhagen, Ron Brightwell, and Anthony Skjellum. "Finepoints: Partitioned multithreaded MPI communication". In: *International Conference on High Performance Computing*. Springer. 2019, pp. 330–350.

[19] Taylor Groves, Yizi Gu, and Nicholas J Wright. "Understanding performance variability on the aries dragonfly network". In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2017, pp. 809–813.

[20] Jahanzeb Maqbool Hashmi, Shulei Xu, Bharath Ramesh, Mohammadreza Bayatpour, Hari Subramoni, and Dhabaleswar K DK Panda. "Machine-agnostic and Communication-aware Designs for MPI on Emerging Architectures". In: *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2020, pp. 32–41.

[21] Katrin Heitmann, Hal Finkel, Adrian Pope, Vitali Morozov, Nicholas Frontiere, Salman Habib, Esteban Rangel, Thomas Uram, Danila Korytov, Hillary Child, et al. "The outer rim simulation: A path to many-core supercomputers". In: *The Astrophysical Journal Supplement Series* 245.1 (2019), p. 16.

[22] Karl Scott Hemmert, Ray Bair, Abhinav Bhatale, Taylor Groves, Nikhil Jain, Cannada Lewis, Misbah Mubarak, Scott D Pakin, Robert Ross, and Jeremiah J Wilke. "Evaluating Trade-offs in Potential Exascale Interconnect Technologies". In: (2020).

[23] Michael A Heroux, Douglas W Doerfler, Paul S Crozier, James M Willenbring, H Carter Edwards, Alan Williams, Mahesh Rajan, Eric R Keiter, Heidi K Thornquist, and Robert W Numrich. "Improving performance via mini-applications". In: *Sandia National Laboratories, Tech. Rep. SAND2009-5574* 3 (2009).

[24] Sascha Hunold and Alexandra Carpen-Amarie. "Reproducible MPI benchmarking is still not as easy as you think". In: *IEEE Transactions on Parallel and Distributed Systems* 27.12 (2016), pp. 3617–3630.

[25] *IBM Spectrum MPI, Accelerating high-performance application parallelization.* `https://www.ibm.com/products/spectrum-mpi`. Accessed: 2022-05-01.

[26] Emmanuel Jeannot, Farouk Mansouri, and Guillaume Mercier. "A hierarchical model to manage hardware topology in MPI applications". In: *Proceedings of the 24th European MPI Users' Group Meeting*. 2017, pp. 1–11.

[27] Yao Kang, Xin Wang, and Zhiling Lan. "Q-adaptive: A Multi-Agent Reinforcement Learning Based Routing on Dragonfly Network". In: *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*. 2020, pp. 189–200.

[28] Yao Kang, Xin Wang, Neil McGlohon, Misbah Mubarak, Sudheer Chunduri, and Zhiling Lan. "Modeling and Analysis of Application Interference on Dragonfly+". In: *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 2019, pp. 161–172. ISBN: 9781450367233.

[29] Fulya Kaplan, Ozan Tuncer, Vitus J Leung, Scott K Hemmert, and Ayse K Coskun. "Unveiling the interplay between global link arrangements and network management algorithms on dragonfly networks". In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE. 2017, pp. 325–334.

[30] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. "Technology-driven, highly-scalable dragonfly topology". In: *2008 International Symposium on Computer Architecture*. IEEE. 2008, pp. 77–88.

[31] Wilhelm Kirch. "Pearson's correlation coefficient". In: *Encyclopedia of Public Health* (2008), pp. 1090–1091.

[32] Pouya Kousha, Kamal Raj Sankarapandian Dayala Ganesh Ram, Mansa Kedia, Hari Subramoni, Arpan Jain, Aamir Shafi, Dhabaleswar Panda, Trey Dockendorf, Heechang Na, and Karen Tomko. "INAM: Cross-stack Profiling and Analysis of Communication in MPI-based Applications". In: *Practice and Experience in Advanced Research Computing*. 2021, pp. 1–11.

[33] Yuanlai Liu, Zhengchun Liu, Rajkumar Kettimuthu, Nageswara Rao, Zizhong Chen, and Ian Foster. "Data Transfer between Scientific Facilities – Bottleneck Analysis, Insights and Optimizations". In: *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 2019, pp. 122–131.

[34] *Marconi100, The new accelerated system.* URL: `https://www.hpc.cineca.it/hardware/marconi100` (visited on 05/01/2022).

[35] Neil McGlohon, Christopher D Carothers, K Scott Hemmert, Michael Levenhagen, Kevin A Brown, Sudheer Chunduri, and Robert B Ross. "Exploration of Congestion Control Techniques on Dragonfly-class HPC Networks Through Simulation". In: *2021 International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE. 2021, pp. 40–50.

[36] George Michelogiannakis, Khaled Z Ibrahim, John Shalf, Jeremiah J Wilke, Samuel Knight, and Joseph P Kenny. "Aphid: Hierarchical task placement to enable a tapered fat tree topology for lower power and cost in hpc networks". In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE. 2017, pp. 228–237.

[37] Md Atiqul Mollah, Peyman Faizian, Md Shafayat Rahman, Xin Yuan, Scott Pakin, and Mike Lang. "A comparative study of topology design approaches for HPC interconnects". In: *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE. 2018, pp. 392–401.

[38] Md Atiqul Mollah, Wenqi Wang, Peyman Faizian, MD Shafayat Rahman, Xin Yuan, Scott Pakin, and Michael Lang. "Modeling Universal Globally Adaptive Load-Balanced Routing". In: *ACM Trans. Parallel Comput.* 6.2 (Aug. 2019).

[39] Javier Navaridas, Josh Lant, Jose A Pascual, Mikel Lujan, and John Goodacre. "Design Exploration of Multi-tier Interconnection Networks for Exascale Systems". In: *Proceedings of the 48th International Conference on Parallel Processing*. 2019, pp. 1–10.

[40] Md Nahid Newaz, Md Atiqul Mollah, Peyman Faizian, and Zhou Tong. "Improving adaptive routing performance on large scale Megafly topology". In: *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE. 2021, pp. 406–416.

[41] *OSU Micro-Benchmarks 5.8*. Aug. 2021. URL: `https://mvapich.cse.ohio-state.edu/benchmarks/`.

[42] Marcelo Ponce, Ramses van Zon, Scott Northrup, Daniel Gruner, Joseph Chen, Fatih Ertinaz, Alexey Fedoseev, Leslie Groer, Fei Mao, Bruno C Mundim, et al. "Deploying a top-100 supercomputer for large parallel workloads: The Niagara Supercomputer". In: *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning)*. 2019, pp. 1–8.

[43] *POWER9 processor chip*. URL: `https://www.ibm.com/it-infrastructure/power/power9`.

[44] Md Shafayat Rahman, Saptarshi Bhowmik, Yevgeniy Ryasnianskiy, Xin Yuan, and Michael Lang. "Topology-Custom UGAL Routing on Dragonfly". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '19. Denver, Colorado: Association for Computing Machinery, 2019. ISBN: 9781450362290.

[45] Jose Rocher-Gonzalez, Jesus Escudero-Sahuquillo, Pedro J Garcia, Fransico J Quiles, and Gaspar Mora. "Efficient Congestion Management for High-Speed Interconnects using Adaptive Routing". In: *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE. 2019, pp. 221–230.

[46] Amit Ruhela, Shulei Xu, Karthik V Manian, Hari Subramoni, and Dhabaleswar K Panda. "Analyzing and Understanding the Impact of Interconnect Performance on HPC, Big Data, and Deep Learning Applications: A Case Study with InfiniBand EDR and HDR". In: *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2020, pp. 869–878.

[47] Alexander Shpiner, Zachy Haramaty, Saar Eliad, Vladimir Zdornov, Barak Gafni, and Eitan Zahavi. "Dragonfly+: Low cost topology for scaling datacenters". In: *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*. IEEE. 2017, pp. 1–8.

[48] David Skinner and William Kramer. "Understanding the causes of performance variability in HPC workloads". In: *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005*. IEEE. 2005, pp. 137–149.

[49] *Slurm, Slurm's job allocation policy for dragonfly network*. 2021. URL: `https://github.com/SchedMD/slurm/blob/master/src/plugins/select/linear/select_linear.c`.

[50] Staci A Smith, Clara E Cromey, David K Lowenthal, Jens Domke, Nikhil Jain, Jayaraman J Thiagarajan, and Abhinav Bhatele. "Mitigating inter-job interference using adaptive flow-aware routing". In: *SC18: International Conference for*

*High Performance Computing, Networking, Storage and Analysis*. IEEE. 2018, pp. 346–360.

[51] Hari Subramoni, Xiaoyi Lu, and Dhabaleswar K Panda. "A scalable network-based performance analysis tool for MPI on large-scale HPC systems". In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2017, pp. 354–358.

[52] Kaushik Kandadi Suresh, Bharath Ramesh, Seyedeh Mahdieh Ghazimirsaeed, Mohammadreza Bayatpour, Jahanzeb Hashmi, and Dhabaleswar K Panda. "Performance Characterization of Network Mechanisms for Non-Contiguous Data Transfers in MPI". In: *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2020, pp. 896–905.

[53] Wei Tang, Narayan Desai, Daniel Buettner, and Zhiling Lan. "Analyzing and adjusting user runtime estimates to improve job scheduling on the Blue Gene/P". In: *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE. 2010, pp. 1–11.

[54] Min Yee Teh, Jeremiah J Wilke, Keren Bergman, and Sebastien Rumley. "Design space exploration of the dragonfly topology". In: *International Conference on High Performance Computing*. Springer. 2017, pp. 57–74.

[55] Yiltan Hassan Temuçin, Amir Hossein Sojoodi, Pedram Alizadeh, Benjamin Kitor, and Ahmad Afsahi. "Accelerating Deep Learning Using Interconnect-Aware UCX Communication for MPI Collectives". In: *IEEE Micro* 42.2 (2022), pp. 68–76.

[56] *Top500, MARCONI-100*. `https://www.top500.org/system/179845/`. Accessed: 2022-05-01.

[57] Xin Wang, Misbah Mubarak, Yao Kang, Robert B. Ross, and Zhiling Lan. "Union: An Automatic Workload Manager for Accelerating Network Simulation". In: *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2020, pp. 821–830.

[58] Xin Wang, Misbah Mubarak, Xu Yang, Robert B Ross, and Zhiling Lan. "Trade-off study of localizing communication and balancing network traffic on a dragonfly system". In: *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2018, pp. 1113–1122.

[59] Xin Wang, Xu Yang, Misbah Mubarak, Robert B Ross, and Zhiling Lan. "A Preliminary Study of Intra-Application Interference on Dragonfly Network". In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2017, pp. 643–644.

[60] Ke Wen, Payman Samadi, Sebastien Rumley, Christine P Chen, Yiwen Shen, Meisam Bahadori, Keren Bergman, and Jeremiah Wilke. "Flexfly: Enabling a reconfigurable dragonfly through silicon photonics". In: *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2016, pp. 166–177.

[61] Jeremiah J Wilke and Joseph P Kenny. "Opportunities and limitations of Quality-of-Service in Message Passing applications on adaptively routed Dragonfly and Fat Tree networks". In: *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2020, pp. 109–118.

[62] Andy B Yoo, Morris A Jette, and Mark Grondona. "Slurm: Simple linux utility for resource management". In: *Workshop on job scheduling strategies for parallel processing*. Springer. 2003, pp. 44–60.

[63] Felix Zahn and Holger Fröning. "On network locality in mpi-based hpc applications". In: *49th International Conference on Parallel Processing-ICPP*. 2020, pp. 1–10.

[64] Jerrold H Zar. "Spearman rank correlation". In: *Encyclopedia of biostatistics* 7 (2005).

[65] Yijia Zhang, Taylor Groves, Brandon Cook, Nicholas J Wright, and Ayse K Coskun. "Quantifying the impact of network congestion on application performance and network metrics". In: *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2020, pp. 162–168.

[66] Yijia Zhang, Ozan Tuncer, Fulya Kaplan, Katzalin Olcoz, Vitus J Leung, and Ayse K Coskun. "Level-spread: A new job allocation policy for dragonfly networks". In: *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2018, pp. 1123–1132.

[67] Zhou Zhou, Xu Yang, Zhiling Lan, Paul Rich, Wei Tang, Vitali Morozov, and Narayan Desai. "Improving batch scheduling on blue gene/q by relaxing 5d torus network allocation constraints". In: *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE. 2015, pp. 439–448.