The Journal of Supercomputing
https://doi.org/10.1007/s11227-024-06040-w

Analysis and prediction of performance variability in large-scale computing systems

Majid Salimi Beni[1] · **Sascha Hunold**[2] · **Biagio Cosenza**[1]

Accepted: 3 March 2024

Abstract
The development of new exascale supercomputers has dramatically increased the need for fast, high-performance networking technology. Efficient network topologies, such as Dragonfly+, have been introduced to meet the demands of data-intensive applications and to match the massive computing power of GPUs and accelerators. However, these supercomputers still face performance variability mainly caused by the network that affects system and application performance. This study comprehensively analyzes performance variability on a large-scale HPC system with Dragonfly+ network topology, focusing on factors such as communication patterns, message size, job placement locality, MPI collective algorithms, and overall system workload. The study also proposes an easy-to-measure metric for estimating network background traffic generated by other users, which can be used to estimate the performance of our job accurately. The insights gained from this study contribute to improving performance predictability, enhancing job placement policies and MPI algorithm selection, and optimizing resource management strategies in supercomputers.

Keywords High performance interconnects · Performance variability · MPI · Dragonfly+ topology · Performance predictability

✉ Majid Salimi Beni
msalimibeni@unisa.it

Sascha Hunold
hunold@par.tuwien.ac.at

Biagio Cosenza
bcosenza@unisa.it

[1] Department of Computer Science, University of Salerno, Salerno, Italy

[2] Faculty of Informatics, TU Wien, Vienna, Austria

Published online: 28 March 2024

1 Introduction

This text will be highlighted. In recent years, with the development of Exascale computing systems and the desire to add more computation resources and nodes to the clusters, the gap between computation and communication has become wider. Recent high performance computing (HPC) and distributed applications require high amounts of computational resources, leading to the development of supercomputers with many nodes [1–3]. Adding more nodes and reaching Exascale, however, is challenging, and there is a need to have efficient software tools [4, 5], high-performance network interconnects [6] and topologies that offer high bandwidth and low latency [7, 8], and therefore higher performance. Hence, modern HPC systems use efficient intra/inter-node interconnects [9] such as NVlink [10] and Infiniband [11], and topologies like Dragonfly [12], Torus [13], and Fat-tree [14]. Employing an efficient network technology can also contribute to improving performance predictability in such large-scale systems [15].

Network topologies play a crucial role in the overall performance of supercomputers, determining how the nodes are connected and how the data is transmitted. Efficient network topologies enable faster communication and facilitate higher levels of parallelism, contributing to higher resource utilization, better scalability, energy efficiency, performance predictability, and fault tolerance [16, 17]. Therefore, recent topologies aim to develop novel network architectures that meet the increasing demands for computational power and data-intensive applications. Well-designed network topologies can minimize communication bottlenecks and reduce latency, leading to more reliable and consistent execution times for various computational tasks. As a result, advancements in network architectures are continually sought to enhance performance predictability and meet the growing requirements of modern computing workloads.

Dragonfly-based topologies are widely used, and many of the top supercomputers worldwide are developed based on these topologies [18]. Dragonfly+ [19] has recently been introduced as an improved version of Dragonfly, which offers better network utilization, scalability, and router buffer utilization. However, despite these enhancements, it still suffers performance variability. The variability affects system and application performance, making it crucial for the batch scheduler to have a more precise estimate of application runtime to make accurate scheduling decisions [20, 21].

Performance variability is the fluctuation in a single program's performance over different executions, and the program's performance deviates from its expected or average behavior. Performance variability can arise due to factors such as system load, resource contention, hardware characteristics, software design, or environmental conditions [22, 23]. It can impact the predictability, reliability, and overall quality of a system or application and can be a critical concern in supercomputers. Since supercomputers often execute highly parallel and demanding computational tasks, even slight variations in their performance can significantly affect the accuracy and efficiency of simulations, data analysis, or scientific computations. Load imbalance, network congestion, system noise, and

communication patterns are among the most influential factors making the performance varying. It has been, however, shown that the performance variability is primarily caused by the network-related elements [15, 24–26].

In large-scale systems, network elements such as routers and links are the resources that, unlike the computation units, cannot be exclusively used by a single user. Large-scale clusters are usually used by many users simultaneously, each with different utilization patterns in terms of program workflow, number of nodes, and data communication. Performance variability may arise when multiple users or jobs compete for limited network bandwidth and resources. This may cause network congestion, decrease the quality of service, and make the network performance unpredictable since a significant share of each MPI communication primitive is spent while transferring the data [27]. Especially when multiple users have varying communication patterns, different jobs may require different amounts of network bandwidth, resulting in imbalances and potential performance degradation, impacting the performance of other jobs sharing the same network elements. Recent studies have been focusing on addressing these issues and providing more performance predictability through the monitoring, prediction, and balancing of network traffic [28–31], and taking into consideration the topological and network design aspects [32–35]. Performance predictability is crucial in supercomputers as it enables efficient resource allocation, supports workflow management, ensures scientific reproducibility, facilitates real-time applications, maintains Quality of Service (QoS), and improves user satisfaction [36].

This study investigates performance variability in a large-scale compute cluster featuring a Dragonfly+ topology. The analysis focuses on several factors known to contribute to performance variability. We study the influence of various collective operations (Broadcast, Reduce, and Alltoall), different message sizes, the localities of job placement by the job scheduler (SLURM), MPI collective algorithms (provided through Open MPI), and the effects of system workload. As part of our study, we tackle the challenging measurement of network background traffic generated by other users. To address this issue, we propose an easy-to-measure and low-cost metric that uses the information provided by the job scheduler and estimates such utilization. Additionally, we highlight the effect of the routing strategy on communication performance in this cluster. By analyzing these aspects, we aim to comprehensively understand performance variance in a supercomputer and provide some hints to predict the performance.

2 Motivation and contributions

In this section, we explain the motivations behind the paper and present our contributions. Figure 1 shows the latency distribution of running Broadcast 100 times, three times a day on different days and at different hours. Although most of the latencies happen in less than 0.2 ms, some runs take much longer, and their average latency takes up to 1.3 ms. From this figure, it is clear there is high variability in the runtime of a communication benchmark, and a fraction of runs are taking longer than the majority. Unlike Fig. 1 where each point on the plot represents the mean of

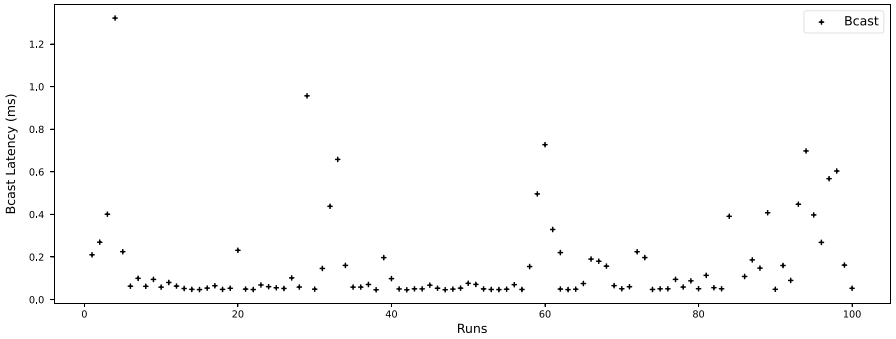


Fig. 1 The runtime distribution of running a Broadcast for 100 times on different days, with 10KB data, on 16 nodes (32 processes per node and 512 processes overall). There is no restriction on the node allocation; the allocated nodes may have changed from run to run. Each point is the arithmetic mean of 1000 iterations of running the benchmark

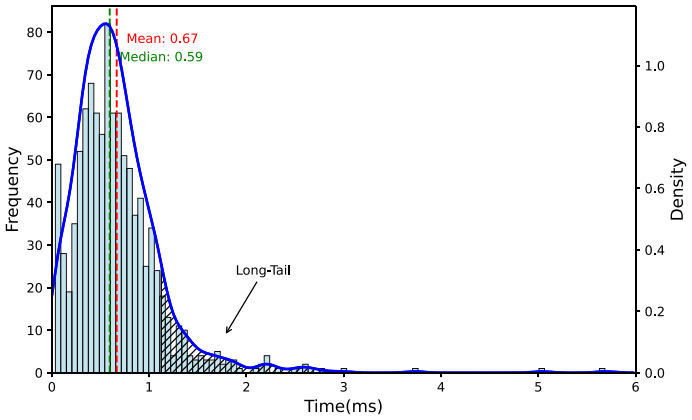


Fig. 2 Long-tail (90th percentile) of the latency distribution of 1000 iterations of Broadcast with 10KB data on 512 processes on Dragonfly+

running the benchmark on different days, Fig. 2 shows the distribution of a single run and represents the distribution of 1000 iterations of that run. Interestingly, the data are not distributed symmetrically (e.g., not Gaussian), and the density distribution is skewed-shaped, causing a gap between the mean and median. In this figure, although most of the latencies are spread around the median, almost 15% of them take longer than the 90th percentile (the hatched area), indicating that the distribution has a long tail. This long tail in the distribution has an adverse effect on the overall performance, resulting in highly unpredictable job execution.

The performance variability is mainly associated with several network and communication-related aspects; this article aims to study and identify the main reasons behind such differences in performance and provide more insights into performance predictability in such clusters and topologies. This work focuses on the Dragonfly+ topology, which is one of the most popular topologies among the top

supercomputers [37, 38], and has shown a better network utilization than Dragonfly [19].

To our knowledge, this study is the first comprehensive analysis of performance variability in a Dragonfly+-based supercomputer. Unlike previous research that relies on simulated environment [39, 40], our approach leverages real-world data extracted from a large-scale compute cluster. The insights of this work can be used to model the cluster's workload occupancy and provide valuable feedback to improve performance predictability by enhancing job placement policies and resource management strategies and, overall, improving the performance of a supercomputer.

2.1 Contributions

The contributions of this work, which is an extension of our previous work [41, 42], include a comprehensive study of performance variability on a real-world supercomputer with Dragonfly+ topology, an easy-to-measure and low-cost metric to estimate the cluster's workload based on the information available at the job submission time, an analysis of the impact of routing strategy and the MPI collective algorithm on the performance variability, and further evaluation of two real-world communication-intensive applications, HACC and miniAMR.

The remaining sections of the article are structured as follows: Sect. 3 provides an overview of related work in the field. Section 4 details the experimental setup used in the study. Section 5 presents the analysis of the latency distribution. Section 6 describes our approach for measuring background traffic. Section 7 offers further analysis of the measured background traffic, and lastly, Sect. 8 provides the discussion and concludes the article.

3 Related work

Since the development of supercomputers, performance predictability has always been of attention, and many researchers have studied the performance variance in such clusters and tried to mitigate this problem. In many studies, network-related elements are investigated as the main reason for performance variability in large-scale clusters [43–46].

Application-related variability studies Some research has investigated the application itself to identify which characteristics in an application can potentially make the performance variable. Zahn et al. [47] explored how communication patterns of an application and their mapping to the topology may vary the performance. Micro-benchmarking [48, 49], program profiling [50], stack tracing [51], performance modeling [52], and studying the collective communications [53] have been other approaches to detect the variance in an application's runtime on a cluster. Taking into account the internal characteristics of the program is, however, not enough to have an accurate understanding of the variance in its performance. Running the program itself or doing benchmarking is time-consuming and imposes high overhead. Besides, in online performance variability detection, in which the external condition

changes and the analysis should be done in different time frames to have updated information, these methods may discard time sequence information, which reduces the accuracy of the variance detection.

Studies considering external factors In some works, the external factors that can potentially impact job performance have been taken into account. Routing strategies [54, 61–65], network designs [66–68], congestion and interference [55, 56, 69], background traffic [57], and job allocation strategies [70–74] are among the studied characteristics that can make the performance variable. Monitoring the cluster and collecting detailed information from the network can provide a good indicator to identify the performance behavior of the running jobs on the cluster. However, this information is not always accessible to the users, and root access is usually needed to perform such monitoring. Moreover, these methods are not generally portable to other clusters since clusters may have different hardware vendors with different profiling tools and counters. Accordingly, many of these methods rely on simulated data, which might not reflect the exact behavior of a real-world supercomputer. Also, not considering the application's behavior, such as its communication intensity and communication patterns, may lead to inaccurate performance prediction.

Mixed approaches Due to the inaccuracies associated with the aforementioned methods, some studies have considered both the program's internal and external characteristics. Application-related attributes such as MPI communications, I/O traffic, and message size, together with job allocation policies, are explored by Brown et al. [58]. Bhatele et al. [24] performed application profiling and network traffic analysis to understand the performance variability better. Other studies also considered program code analysis [59, 60] while profiling network counters or doing offline analyses. Although these studies can better predict performance variance, they still have some problems of the studies, which consider external factors only. Moreover, offline analysis done in some of these works may not reflect the transient behavior of a supercomputer.

Table 1 compares our approach with the related work on different parameters in predicting and analyzing performance variability. Our work is shown with bold.

4 Experimental setup

The current study was conducted on a high-performance computing cluster, Marconi100, located at the CINECA supercomputing center [18].

4.1 Computing and network

The Marconi100 cluster comprises 980 compute nodes (plus the login nodes), each equipped with two IBM POWER9 AC922 processors with 16 cores running at 2.6 (3.1 turbo) GHz, four NVIDIA Volta V100 GPUs with 16GB, and 256 GB of memory per node. In total, the cluster has 347,776 CPU cores and 347,776 GB of memory.

Table 1 Comparison of different performance variability studies

Work	Method/Approach	Considered Parameters
Hoefler et al. [48]	Benchmarking, simulation-based	System noise, collective operations
Maricq et al. [49]	Benchmarking, statistical methods	Benchmark performance data
Smith et al. [54]	Benchmarking	Network counters, routing, topology
McGlohon et al. [55]	Simulation-based	Routing, congestion
Shah et al. [56]	Application profiling	Profile output, communication and I/O features
Zhang et al. [57]	Benchmarking, network counters profiling	Network congestion, node allocation
Wang et al. [28]	Simulation-based	Node allocation, application performance data
Brown et al. [58]	Simulation-based	I/O, message sizes, communication intervals, job sizes
Tang et al. [59]	On-line compiler-based	Static code features, application performance
Zheng et al. [60]	Compiler-based, workload analysis, statistical analyses	Static code features, memory, IO
Our work	**Heuristic-based, statistical analyses**	**Node allocation, communication patterns, message size, collective algorithms, system workload, routing**

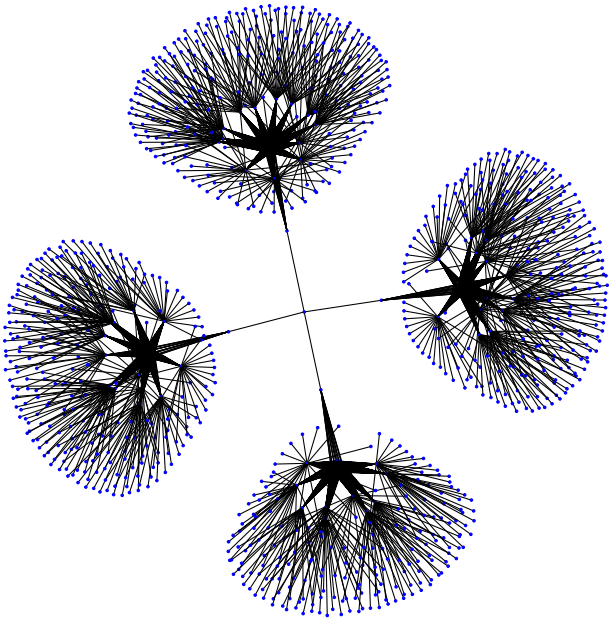


Fig. 3 The Dragonfly+ schematic implemented in Marconi100. The blue points show the compute nodes and intermediate network switches (color figure online)

The internal interconnect of Marconi100 is a Mellanox InfiniBand EDR Dragonfly+. This Dragonfly+ implemented in this cluster, as shown in Fig. 3, consists of four large groups of nodes called "islands" and smaller groups of nodes within each island connected to one switch called "groups." The main difference between Dragonfly and Dragonfly+ is that in Dragonfly+, intra-island routers are connected as a bipartite graph to enhance scalability.

4.2 Software, microbenchmarks, and applications

The operating system of the machines is Red Hat Enterprise 7.6, IBM Spectrum-MPI 10.4 and OpenMPI 4.1.4 are installed on the cluster, and SLURM 21.08 is used for resource management. Adaptive Routing [75] is the default routing strategy used to prevent link contention and handle hardware failures.

The analyses are done using the OSU microbenchmarks [76], and we have used three communication microbenchmarks (Broadcast, Reduce, and Alltoall) as well as two real-world applications (HACC [77] and miniAMR [78]). As suggested by [79], each collective is executed in 1-millisecond intervals 1000 times inside a loop to show the performance variance. Furthermore, in all the experiments, we allocate 16 nodes on the cluster (We cannot go beyond due to the limitations of our accounts). The data shown in the article have been collected over 3 months, encompassing different cluster utilizations.

5 Analysis of performance variability

In this section, we perform a latency analysis to highlight the performance variance problem on this Dragonfly+-based cluster. For this purpose, we indicate the impact of the locality of node allocation on different collective communications and show how node allocation can push the tail of the latency distribution.

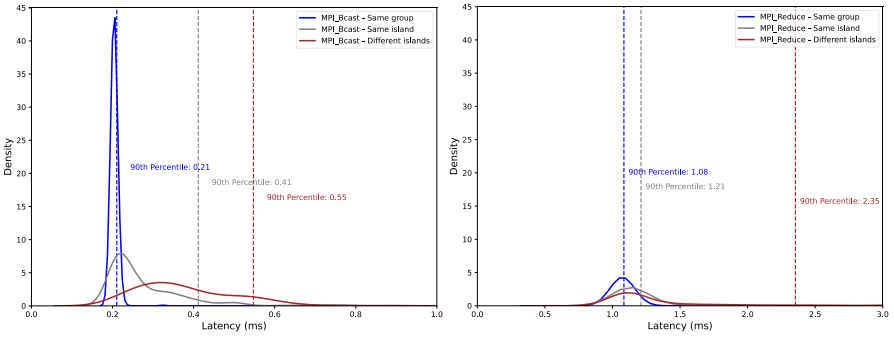
5.1 The impact of node allocation on performance variance

This section shows how node allocation (job placement) affects performance variance. Considering the topology shown in Fig. 3, we define the three following locality levels, sorted from highest to lowest locality:

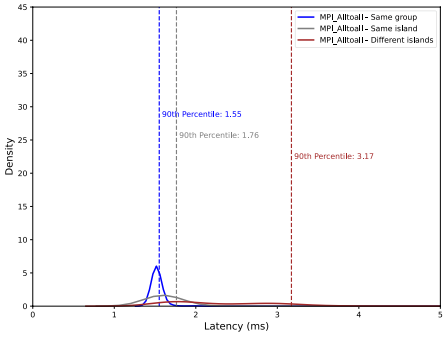
1. Same Group: All the nodes are allocated within a single group; only a single network switch connects every two nodes. This case exposes the most locality.
2. Same Island: Nodes are allocated across different groups of a single island.
3. Different Islands: There is no limitation on node allocation, and they are distributed on all the islands and different groups; this scenario imposes less locality than the previous scenarios.

The latency distribution of three collectives with three node allocation levels is depicted in Fig. 4. We show the 90th percentile for each distribution as an indicator of the tail of the distributions. The different islands' experiments are performed at the same time, one after the other, to have their runtimes with similar network conditions. The observations from this figure are:

- The Broadcast (Fig. 4a) exhibits the best performance for all the allocation strategies compared to the other collectives, with the shortest tail between three collectives (notice, the x-axes being in different scales in the three plots). For the three locality levels, Broadcast shows less variance than the correspondings in Reduce and Alltoall.
- For Broadcast, the peaks of different islands and the same island are 8 and 4, respectively, and they possess a peak much lower than the same group (44). This collective significantly benefits local communications within the same group, demonstrating lower latency (mean time = 0.20 ms) and shorter tail (90th percentile = 0.21 ms).
- The Reduce (Fig. 4b) demonstrates similar mean latency for the same group and same island allocations, with values of approximately 1.17 ms and 1.18 ms, respectively. The distribution of these communication times does not exhibit a significant long tail. However, when considering different islands, the 90th percentile is 2.45 ms, and the distribution extends further, with a tail reaching up to 10 ms (not shown in the plot).
- The Alltoall collective (Fig. 4c) demonstrates the slowest and most variable performance, mainly when all the nodes are allocated on different islands. The



(a) Broadcast (b) Reduce



(c) Alltoall

Fig. 4 Latency frequency distribution of the three collectives with 1 MB message size, repeated for 1000 iterations, with the three allocation locality levels on 16 nodes, one process per node to emphasize only the inter-node latency

frequency distribution of communication times reveals a very long tail, with the maximum observed communication time reaching 13 ms and the peak of the distribution of 1.

In general, allocating all nodes to the same group has shown benefits for collective communications. There are, however, two barriers making us unable to allocate all the required nodes to the same group. First, the number of nodes within each group in Dragonfly+ is limited (20 nodes in Marconi100), and the job may require more than available nodes in each group. Second, since many users utilize the cluster at the same time, some nodes within the groups may have already been allocated by their jobs, and the job scheduler has to wait a long time to find idle nodes within the same group for our jobs. In our experiments, the maximum waiting time to allocate 16 nodes in the same group by SLURM has been around 21 days.

By default, SLURM [80] attempts to allocate jobs to idle nodes within the same group unless the user specifies specific nodes in the host file or modifies the allocation policy. However, due to the limited number of available idle nodes in the same group, the job scheduler searches for switches (groups) with the fewest

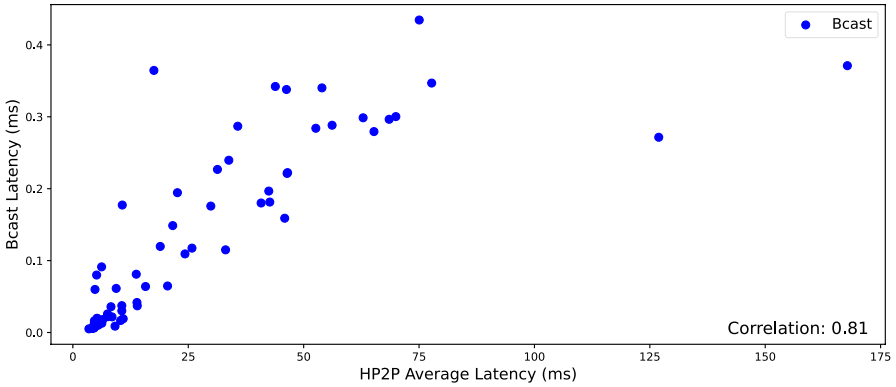


Fig. 5 The correlation between latencies of HP2P and Broadcast on 16 nodes, one process per node, and 10KB of message size. HP2P is performed 1000 iterations with 4KB

idle nodes and assigns the nodes connected to these switches. This process is repeated until all requested nodes are assigned. Therefore, based on the requested node count and the availability of idle nodes in the cluster, SLURM may allocate jobs to nodes in different groups within the same island or across multiple islands. Our observations suggest that the latter scenario is more likely due to the constraints imposed by the availability of idle nodes. Therefore, in the rest of the paper, no node allocation restriction has been specified for all the experiments, and the nodes are allocated on different islands by default.

6 Predicting variability

As suggested by the related work [48, 57], it is essential to monitor the external conditions on the cluster alongside considering the application's communication patterns. In real-world supercomputers, users do not have exclusive access to dedicated systems; instead, multiple users submit jobs concurrently. While computing nodes can be allocated exclusively (by default, they are shared), the network is a shared resource and, therefore, inherently subject to contention. As the number of active jobs increases, more nodes will be allocated, and consequently, the network congestion can potentially increase due to the heavier competition for network resources. This section examines the impact of background traffic generated by other users' jobs on performance variability. We propose a heuristic that provides an estimate of the workload on the system and then investigate the correlation between this workload and the latency of our job.

6.1 Benchmarking-based estimation

The most straightforward way to estimate the network background traffic and cluster utilization is to measure the bandwidth of the utilized links with a microbenchmark. By correlating the measured bandwidth with the latency of our job, we can estimate

how long the main job takes to be executed. In Fig. 5, we run HP2P microbenchmark [81], which measures the P2P bandwidth and latency between all the pairs involved in the communication by interchanging asynchronous messages, before running the Broadcast. As indicated in the figure, HP2P and Broadcast are 0.81 correlated, and this way can almost accurately predict the runtime of the Broadcast.

This method, however, has some disadvantages and shortcomings. First, by only running a benchmark, we are not collecting any information about the cluster's workload, and it is not clear where the variability originates from and what the external sources of variability are. Second, running a benchmark every time before running a task is an overhead and consumes some computation time. In benchmarking-based methods, there is a trade-off between accuracy and overhead. This means that we need to run the benchmark (HP2P in this case) with an appropriate message size for a certain amount of time, to reach a good accuracy in our estimations. The longer we run the benchmark and the larger the message size we use within the benchmark, the higher the accuracy we will achieve. However, we need to minimize the overhead of running this benchmark by choosing a small message size and running it for fewer iterations while achieving an acceptable accuracy. In this paper, we tested different numbers of iterations and message sizes for HP2P, and the configuration reported (1000 iterations with 4KB) was the point where we observed an acceptable correlation of over 0.8. So, achieving an acceptable accuracy through benchmarking methods takes up to tens of milliseconds, which in our case is even higher than the runtime of the main task (Broadcast operation). In the rest of this article, we propose a zero-overhead method that is also aware of the cluster's workload and can accurately predict the performance.

6.2 Heuristic-based estimation of variability

This section introduces our proposed method to estimate how the cluster's current workload can impact our job's performance. The main source of performance variance is due to external factors, which are mainly the consequences of the activities of other users running their jobs on the cluster simultaneously. Therefore, we try to monitor the cluster's usage by analyzing the job schedule's queue. To that end, we define a new metric, *background network utilization*, which estimates the external workload and network usage by other users' jobs. To collect such information, before running the main job, we query the SLURM's job queue and extract the following information:

- The total number of running jobs.
- The number of nodes allocated by the running jobs.
- The number of nodes shared between the running jobs.
- The total number of cluster's compute nodes (which is constant for each cluster).

Pending jobs, whose scheduling times are uncertain, are not considered in our analysis of background traffic. Additionally, running jobs that allocate only one node are excluded from our calculations since they have minimal communication

with other nodes and thus have no notable network activity. Consequently, our analysis focuses solely on running jobs that allocate at least two nodes. To quantify background traffic, we introduce a heuristic called *background network utilization* (b). This metric is calculated as the number of distinct nodes allocated by running jobs, where each allocation involves at least two nodes, divided by the total number of physical nodes in the cluster. In essence, it represents the proportion of nodes that contribute to communication among all the nodes in the cluster.

The background network utilization (b) ratio is formally defined as

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

where N_c denotes the number of distinct nodes involved in communication within the cluster and N_t the total number of physical compute nodes of the cluster, respectively.

By default, SLURM assigns the nodes to the jobs so that if there are unallocated resources on each node (e.g., GPU), the idle resources can be utilized by other jobs, and hence, those nodes may do more network activities. To account for cases where a node is shared among multiple jobs and to improve the accuracy of the heuristic, we introduce a refinement to the background network utilization metric. In this refined version, we address the presence of shared nodes by counting each shared node multiple times based on their appearance in the allocated nodes by the jobs (that allocate more than two nodes). This approach allows us to give more weight to the nodes that are exploited by more than one job.

As a result, the total number of running nodes may exceed the physical nodes of the cluster (N_t) since we are counting the shared nodes more than once. To incorporate the effect of shared nodes, we introduce a new ratio by dividing the number of nodes contributing to communication (including the shared ones) by the total number of allocated running nodes (including the shared ones). This ratio accounts for the presence of nodes that contribute to different jobs and engage in communication with other nodes. The updated version of the background network utilization, which will be utilized throughout the paper, can be defined as follows:

$$b = \frac{1}{2}\left(\frac{N_c}{N_t} + \frac{N'_c}{N_a}\right), \tag{2}$$

where N'_c denotes the number of nodes involved in the communication, and N_a denotes the total number of allocated running nodes, both accounting for duplicates.

To accurately measure and validate the stability of the background network utilization (b) during the microbenchmark's execution, we incorporate an additional step and query the status of the job scheduler after the execution of the microbenchmark in addition to the previous querying. We compare the two calculated b values, and only if the difference between the two values is below a predefined threshold (5%), we capture the b value as a valid measurement. This approach ensures that the measured b value accurately represents the true utilization and remains consistent throughout the microbenchmark execution.

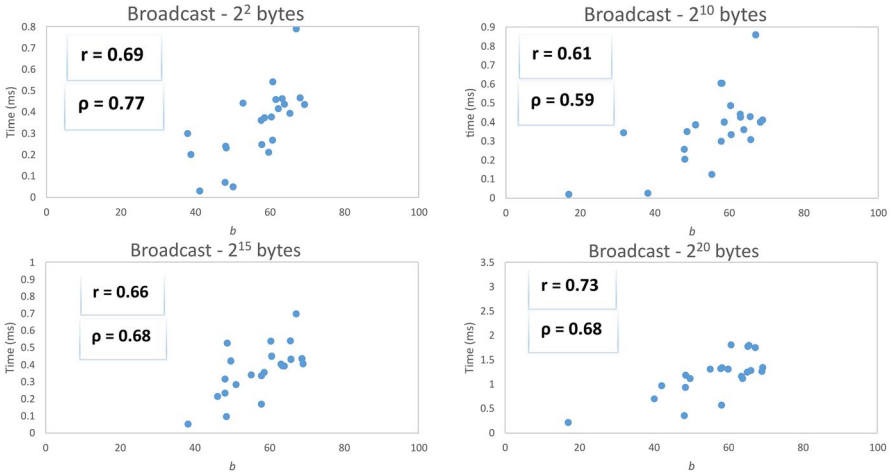


Fig. 6 The correlation between background traffic (b) and the average communication time for Broadcast with different message sizes. The experiments were performed on 16 nodes (1 process per node) allocated on different islands, and each point on the plot represents the average communication time over 1000 iterations

6.3 Correlation analysis

The correlation analysis was conducted to examine the relationship between the background network utilization (b) and communication time, considering various workloads with different data sizes and communication patterns. Two correlation coefficients, Pearson Correlation Coefficient (r) [82] and Spearman Rank Correlation (ρ) [83], were used for this analysis. Pearson's correlation coefficient measures the linear relationship between variables, while Spearman's correlation coefficient assesses the monotonic relationship in the data. In both cases, a value of +1 indicates a strong positive correlation, a value of 0 indicates independence between the variables, and negative values indicate inverse relationships.

Figures 6, 7, and 8 show the relationship between background network utilization (b) and latency for three communication patterns: Broadcast, Reduce, and Alltoall, respectively. Note that point-to-point communication was not explored in this analysis as it was found to be minimally affected by the background traffic. The results presented in this figure demonstrate a strong correlation between the latency of the collectives and the background network utilization metric (b). Moreover, when examining the effect of message size on the correlation between b and communication time, as the message size grows from 2^2, 2^{10}, and 2^{15} to 2^{20} bytes, the correlation between these variables strengthens, as a general trend. This indicates that larger data sizes experience a greater impact from background traffic. The correlations in the Reduce collective for larger data sizes (2^{15} and 2^{20} bytes) are particularly strong compared to other communication patterns. This implies that the communication time in the Reduce collective is heavily influenced by the background traffic, especially when dealing with larger data sizes. Besides, when comparing the

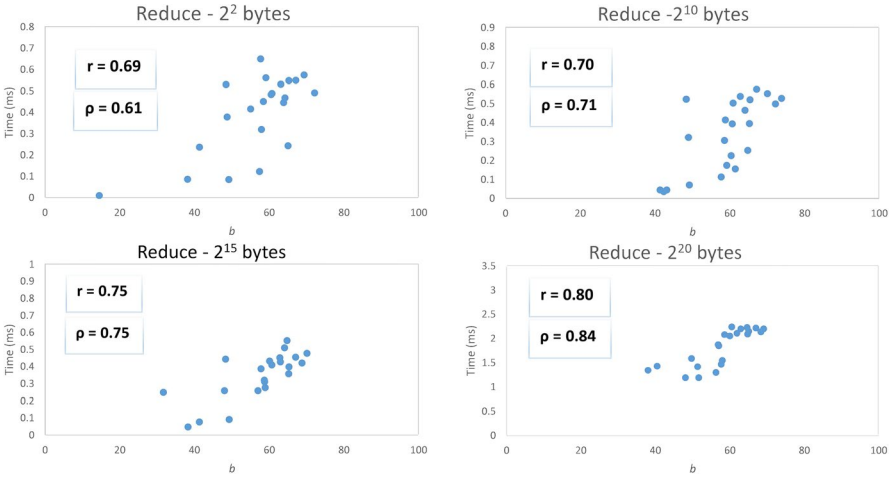


Fig. 7 The correlation between background traffic (b) and the average communication time for Reduce with different message sizes. The experiments were performed on 16 nodes (1 process per node) allocated on different islands, and each point on the plot represents the average communication time over 1000 iterations

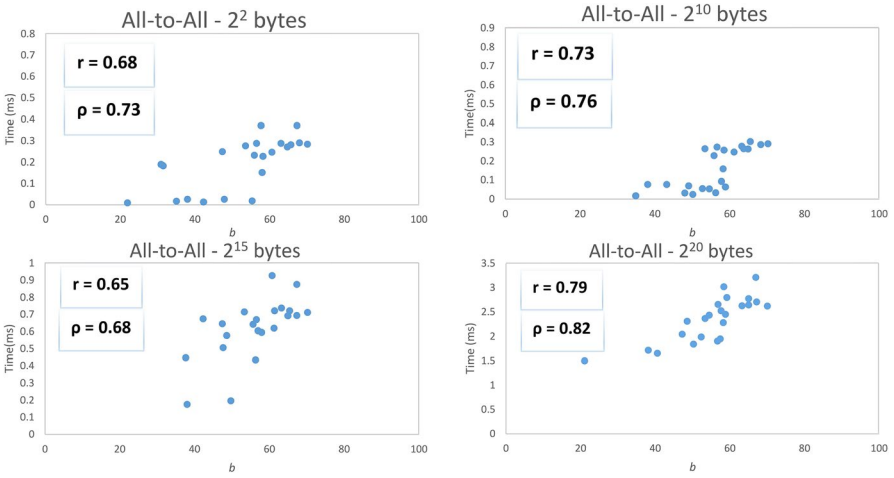
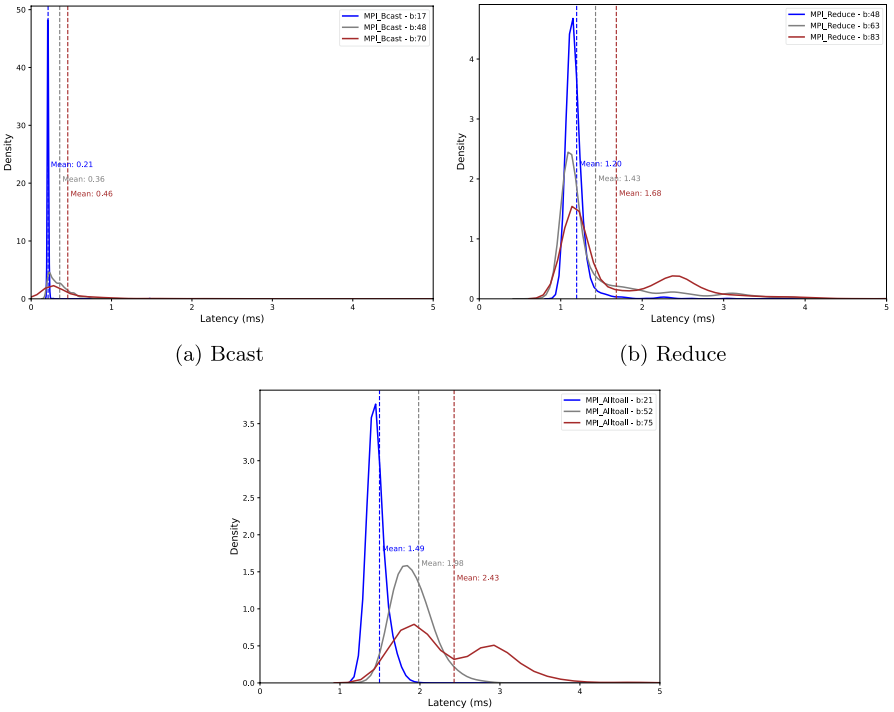


Fig. 8 The correlation between background traffic (b) and the average communication time for Alltoall with different message sizes. The experiments were performed on 16 nodes (1 process per node) allocated on different islands, and each point on the plot represents the average communication time over 1000 iterations

Pearson and Spearman correlations, it can be concluded that the Spearman correlation consistently provides a better fit for our use case. This is due to the monotonic relationship between the two sets of our data in which the variables tend to increase or decrease together without considering the specific magnitude of the change. As



(a) Bcast (b) Reduce

(c) Alltoall

Fig. 9 Frequency distribution of latency of 1000 iterations of Broadcast, Reduce, and Alltoall on 16 nodes (1 process per node) with different background network utilization (b). These nodes are allocated across different islands, and the message size is 2^{20} bytes

shown, it has the tendency to exhibit a stronger correlation between variables, making it a more suitable measure for accurately capturing the relationship between background network utilization and communication time in our experiments.

7 Variability on collectives and applications

So far, we have introduced a heuristic to estimate cluster workload and showed the relation between our metric and the performance of collective communications. In this section, we focus on our definition of performance variability (as the motivation example in Fig. 2) and reveal how the current workload of the cluster impacts the distribution of latency of a single collective when running it multiple times.

Figure 9 illustrates the density distribution of latencies of 1000 iterations of three collectives; This figure focuses on the relationship between background network utilization (b) and the distribution of execution times. To have different b values, the benchmarks were executed on different days. Notice all three plots are on the same

x-axis. Across all three collectives, there is a clear pattern: as the background network utilization increases, the peak of the distribution drops, and the tail becomes longer. This indicates that higher background network utilization leads to more performance variability and degrades communication performance. For the Broadcast collective (Fig. 9a) with $b = 0.17$, the distribution exhibits a high peak, and there is a noticeable gap between the peaks of the higher and lower b values. However, when the background network utilization increases to $b = 0.70$, the corresponding distribution shows an extended tail, indicating highly variable latency ranging from 0.2 ms to 8 ms. It is important to note that the average execution time for 1000 iterations of Broadcast with $b = 0.70$ is up to 6.4 times larger than with $b = 0.17$. This emphasizes the significant impact of background traffic on Broadcast performance. Interestingly, when the nodes are distributed across different islands, and the background traffic is low, Broadcast's performance is comparable to the case where the nodes are allocated to the same group.

Likewise, in Fig. 9b, c, we observe a long-tailed distribution as the background network utilization increases. Particularly for Alltoall, when the background network utilization is high, the tail of the distribution becomes longer, the peak drops and the average latency is much larger. In fact, the mean of the distribution with $b = 0.75$ is approximately 1.6 times larger than with $b = 0.21$. Unlike the other collectives, the Alltoall collective exhibits a significant shift in the peak of the distributions (represented by the median) across different background network utilizations. This shift is attributed to the inherent communication intensity of the All-to-All pattern. In this pattern, all nodes exchange data, leading to a higher volume of data being transmitted through the network. Consequently, the network links become more congested, resulting in a major shift in the peak of the distribution for different traffics.

Furthermore, we observe a bimodal distribution for high background network utilizations in the Reduce and Alltoall collectives. This indicates that a significant number of latencies are mainly happening in 2 time intervals. This behavior is closely related to the Adaptive routing algorithm utilized in this cluster. With Adaptive routing, routers have multiple paths available for each packet transmission. Consequently, some packets follow the shortest (minimal) path, while others traverse alternative, longer (non-minimal) paths. As a result, certain communications experience slower performance compared to the majority due to the penalty associated with selecting the non-minimal path. Figure 9b, c illustrate this phenomenon, where higher background network utilization increases the probability of packets being routed through non-minimal paths, thereby elongating the distribution tail and making it bimodal. It is important to note that the routing strategy cannot be altered in our experiments as we are not using any simulator in our experiments, and changing such policies requires admin access.

Overall, it is evident how background traffic impacts the distributions. While the Adaptive routing strategy helps alleviate the issue to some extent, there are cases where the problem persists, particularly when there is a very high background traffic load. On top of that, when the cluster is highly utilized, and the distributions are bimodal, finding a metric to represent the whole data is a complex task. Neither arithmetic mean nor median is a good representative of the entire distribution as

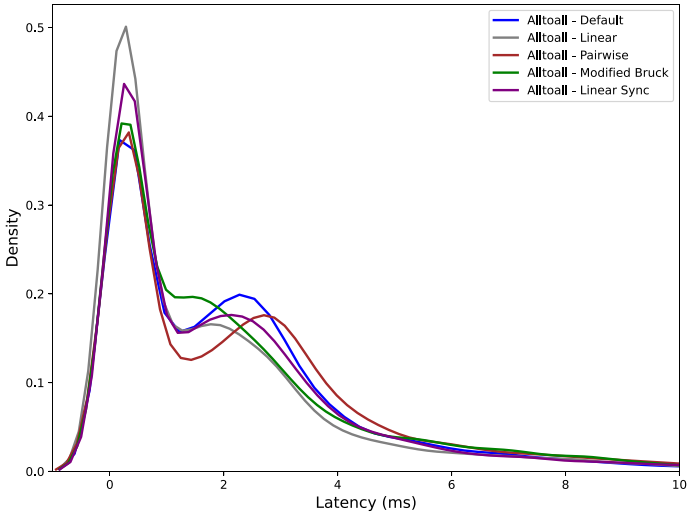


Fig. 10 The latency density distribution of different algorithms of Alltoall implemented in OpenMPI with 512 processes (16 nodes with 32 processes per node), with 4KB data size. Each distribution is representative of 20 runs (each with 1000 iterations) performed on different days. The default algorithm is mapped to Linear Sync in this plot

they do not contain any information regarding the number of peaks of the distribution, and they do not provide accurate details on the tail of the distribution.

7.1 Collective algorithms and latency distributions

In recent MPI implementations, multiple algorithms have been developed for each collective operation, and each algorithm possesses unique internal characteristics, including communication costs and scalability attributes. The selection of an efficient algorithm for MPI collectives is of utmost importance in achieving optimal performance since each algorithm has different scalability, communication overhead, and resource utilization [84–87]. In this section, we highlight how different implementations of each collective can impact its latency distribution. Figure 10 shows the latency distributions of different algorithms of Alltoall. Although all the algorithms deliver the same output, they show pretty different distributions, and unlike the others, Modified Bruck and Linear do not have a completely bimodal distribution.

To better understand, Fig. 11 presents the same data with Violin and Box plots. There are several interesting observations from this figure:

- Besides the difference in the mean and median, the different algorithms have different latency distributions.

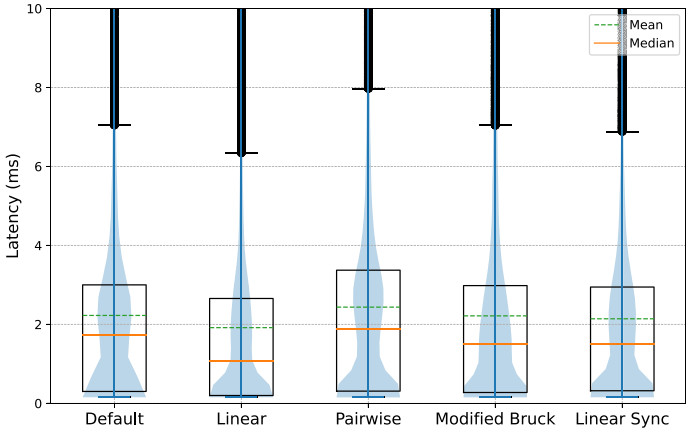


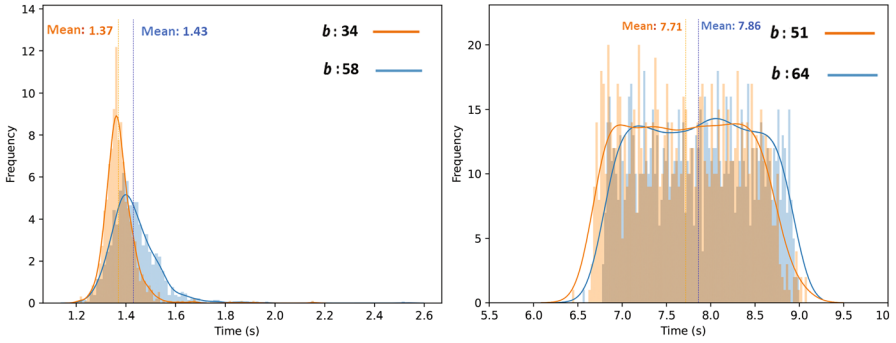
Fig. 11 The latency distribution of different algorithms of Alltoall implemented in OpenMPI with 512 processes (16 nodes with 32 processes per node), with 4KB data size. Each distribution is representative of 20 runs (each with 1000 iterations) performed on different days. The default algorithm is mapped to Linear Sync in this plot

- In addition to the cluster's workload, the chosen collective algorithm can impact the bimodality of the distribution.
- Linear Alltoall algorithm has the shortest tail, and its distribution is not bimodal. Therefore it has the best mean and median. It is probably due to the inherent features of this algorithm, which make it deal better with the packet transmission policy of the Adaptive routing algorithm.
- The Default is mapped to Linear Sync in the OpenMPI selection decision tree, and therefore they not only have similar distribution shape, but the mean, median, and tail of the distribution is also the same for them.

So, apart from the elements discussed in the article, the chosen algorithm for the collective operations can also have an impact on performance variability. However, the behavior of each algorithm may vary with different data sizes and number of nodes (processes). Also, each algorithm may have a different distribution shape while changing the routing strategy. Therefore, it is not possible to find the best algorithm with less variable performance before testing all the possible scenarios.

7.2 Application analysis

Thus far, we have discussed the influence of cluster and network utilization, routing strategy, and the collective algorithm on microbenchmarks latency distribution. In this section, we delve into the effects of background network utilization on two real-world communication-intensive applications to see how does the execution time distribution look like when a program consists of several communication patterns in addition to computation. The chosen applications and their specifications are as follows:



(a) HACC (b) miniAMR

Fig. 12 The frequency and density distributions of 1000 iterations of running HACC and miniAMR applications, considering two different background network utilization on 16 nodes (1 process per node)

- HACC[1]: A cosmology framework designed for performing N-body simulations. It simulates the intricate structure formation process in an expanding space and heavily relies on communication between particles to accurately model gravitational interactions. HACC includes various MPI communication patterns. The number of particles in our experiments is 10 M.
- miniAMR: A mini-application that focuses on stencil calculations performed on a computational domain in the shape of a unit cube. miniAMR is a simplified representation of more complex adaptive mesh refinement (AMR) applications commonly used in computational fluid dynamics (CFD) and other scientific domains. The communication-intensive nature of miniAMR arises from the need to exchange boundary data between neighboring subdomains utilizing several communication patterns. Our test is performed using 4K 3D blocks as input.

In Fig. 12, the network latency distributions for HACC and miniAMR are presented using histograms and density distributions. The HACC distribution, depicted in Fig. 12a, exhibits two distinct distributions. The orange distribution corresponds to a background network utilization (b) of 34, with an average execution time of 1.37s and a peak of 8.9. On the other hand, the blue distribution represents $b = 58$, resulting in an average execution time of 1.43s and a peak latency of 5.2. This indicates that with a 24% increase in b, the average execution time experiences a 4.4% increase. Additionally, both distributions in Fig. 12a have a single bell-shaped curve. Nevertheless, the blue distribution is more dilated, with its tail reaching a latency of 2.5, while the tail of the orange distribution only grows to 2.1. This implies that higher background network utilization leads to a broader range of latencies and potentially longer tails in the distribution.

Nonetheless, for miniAMR, as shown in Fig. 12b, when the background network utilization (b) slightly increases from 51 to 64 with a 13% change, the average execution time rises from 7.71 to 7.86, indicating a 2% increase, and it does not impact so much the distribution shape and just shifts the plot to the right a bit. In contrast to

[1] Hardware/Hybrid Accelerated Cosmology Code.

the previous observations, both plots in this figure exhibit multiple peaks and demonstrate different behaviors, suggesting that the impact of background network utilization on miniAMR differs from HACC. Analyzing the previous study on the two applications [57], it was found that approximately 67% of the overall execution time of HACC is attributed to MPI operations (mostly Allreduce, Scatter, and Gather), while only a negligible fraction (0.1%) is related to blocking collective communications. On the other hand, in the case of miniAMR, Allreduce alone contributed to 9.2% of the overall execution time, and it is the dominant collective pattern. This indicates that miniAMR involves more collective communications with more complex communication patterns.

As shown in this article, the Alltoall and Reduce collectives are more affected by network background traffic. In fact, they exhibit a flatter distribution when exposed to higher network background traffic, and their usage, together with the routing strategy of the cluster, can result in a bimodal distribution in their latency distribution. In the case of miniAMR, an analysis of its code reveals the presence of over 10,000 `MPI_Allreduce` operations that possess a high communication intensity. According to our experience, the latency distribution's shape is mainly dominated by the dominant collective. In Fig. 12b, the distribution of communication latencies becomes flat-topped, primarily attributed to the dominant communication patterns and complexities of the applications compared to microbenchmarks since they also contain computation and their communication part consists of different collective and point-to-point communications with different message sizes. Furthermore, the routing algorithm employed in the cluster plays a role due to the high communication intensity, and a multi-modal distribution is expected as it combines various distributions related to complex communication patterns of the `MPI_Allreduce` operation with the computation times.

8 Discussion and conclusion

In this article, we have analyzed the network latency distribution on a large-scale compute cluster with Dragonfly+ topology and have provided several insights. One notable finding is the significant difference between the performance of the different node allocation strategies; notably, the "same group" allocation policy delivers considerably better performance than the two other policies. When all nodes are allocated to a single group, there is only one hop between any two nodes. Therefore, the minimal and non-minimal paths are the same for Adaptive routing, resulting in a very low impact from the global workload of the cluster. Hence, the latency distribution exhibits a shorter tail and a higher peak and minimizes the effects of background traffic for this allocation strategy. Thus, allocating all required nodes to the same group is advisable if there are enough available idle nodes in that group.

Analyzing the latency distribution based on communication patterns, we have observed that the Broadcast pattern benefits significantly from the locality of job allocation. It exhibits the shortest tail and the higher peak compared to the Reduce and Alltoall, particularly in the same group and same island allocations. However,

when nodes are allocated across different islands, Broadcast is highly affected by the background traffic, resulting in a very long tail compared to cases with lower background traffic. It demonstrates that the communication performance of Broadcast becomes more variable only if the cluster is highly utilized. Nonetheless, the possibility of encountering this situation is rare, as the measured background network utilization has mostly been between 0.40 and 0.70. On the other hand, Alltoall exhibits the most extended tail when there is less locality in job placement. While its distribution is similar to Reduce on the same group, performing Alltoall across different islands leads to a significantly longer tail due to this collective's higher volume of communication.

While allocating the nodes on different islands, our experimental analysis has revealed a two-peak (bimodal) distribution in communication latency, which is attributed to the routing algorithm. This behavior arises from offloading the packets to longer paths while the minimal paths become congested. On top of that, the chosen collective algorithm itself can be another source of performance variability, and its combination with other factors, such as routing, can impact the performance and make it bimodal. Examining the latency distribution of real-world communication-intensive applications, we have observed that the distribution is primarily influenced by its dominant communication pattern. Consequently, as the network background traffic increases, the overall average execution time of the application also tends to increase, and the distribution becomes more skewed.

Above all, when describing the performance of communication-intensive applications on large-scale supercomputers, we showed that neither arithmetic mean nor median can describe the overall performance. They do not retain enough information about the tail of the distribution or the number of peaks. Moreover, when running a distributed application on such systems, if the experiment is not executed enough iterations, the presented execution times may be too discrete and not describe the actual performance.

Regarding the performance variability sources, although there are several factors affecting the performance, such as system activities, MPI, background daemons, file system, and garbage collection, the network-related elements have been proven to be the main sources since they are prone to congestion [22, 88]. Network congestion is not, however, easy to measure and is very challenging in the real world. Network counters available in some clusters can be monitored to estimate network utilization more accurately. Still, these counters are not available in all the clusters, and they are vendor-specific. Moreover, to access such information, admin access to the cluster might be needed. So, in this article, we rely on the data provided by SLURM in user space as the external information alongside the application-related information, which makes our method portable to any cluster that uses SLURM as its job scheduler. However, it should be noted that our estimation of the background network utilization is based on available information from other users and their node allocations, which introduces the possibility of errors since we have no information about their communication patterns, etc. That is why the correlations between our introduced metric and the latency of collectives are not +1 in Figs. 6, 7, and 8.

The b metric introduced in this article can be a good indicator for HPC users to find the right moment to submit their communication-intensive jobs to experience

minimum performance variability and estimate how long their job may take considering the current background traffic on the cluster. According to our findings, the job scheduler needs to consider balancing the node allocation between the jobs allocating single nodes and the jobs that require multiple nodes to avoid congesting links. Moreover, the job scheduler can use the metric introduced in this study to decide when to submit the communication-performing tasks (the ones requiring many nodes) to experience the minimum variability. It also can use this metric to predict the runtime of each of the MPI tasks and further improve its scheduling decisions for other tasks.

Overall, in this article, we tried to analyze and show how the high utilization of the cluster contributes to more competition for shared resources such as the network, causing resource contention and, therefore, making the performance of our job vary from the normal conditions. For future work, we plan to improve the accuracy of our method first by collecting more static and runtime data from the applications and network and then combining our heuristic with regression, statistical, and machine learning techniques. The second future direction is to embed this model into the job scheduler so that it takes care of the utilization of the cluster when making scheduling decisions and, at the right moment, allocates the nodes where performance variability is minimized. The third future work is to design an MPI collective algorithm selection logic that also takes into account the utilization of the cluster and chooses the algorithms that cause less performance variability.

Acknowledgements We acknowledge the CINECA award under the ISCRA initiative, for the availability of high-performance computing resources and support.

Funding Open access funding provided by Università degli Studi di Salerno within the CRUI-CARE Agreement.

References

1. Thoman P, Salzmann P, Cosenza B, Fahringer T (2019) Celerity: high-level C++ for accelerator clusters. In: Euro-Par 2019: Parallel Processing: 25th International Conference on Parallel and Distributed Computing, Göttingen, Germany, August 26–30, 2019, Proceedings 25. Springer, pp 291–303
2. Sojoodi AH, Salimi Beni M, Khunjush F (2021) Ignite-gpu: a gpu-enabled in-memory computing architecture on clusters. J Supercomput 77:3165–3192
3. Bhattacharjee A, Wells J (2021) Preface to special topic: bilding the bridge to the exascale-applications and opportunities for plasma physics. Phys Plasmas 28(9):090401

4. Träff JL, Lübbe FD, Rougier A, Hunold S (2015) Isomorphic, sparse MPI-like collective communication operations for parallel stencil computations. In: Proceedings of the 22nd European MPI Users' Group Meeting, pp 1–10
5. Salzmann P, Knorr F, Thoman P, Cosenza B (2022) Celerity: how (well) does the sycl api translate to distributed clusters? In: International workshop on OpenCL, pp 1–2
6. Temuçin YH, Sojoodi AH, Alizadeh P, Kitor B, Afsahi A (2022) Accelerating deep learning using interconnect-aware UCX communication for MPI collectives. IEEE Micro 42(2):68–76
7. Jain N, Bhatele A, White S, Gamblin T, Kale LV (2016) Evaluating HPC networks via simulation of parallel workloads. In: SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, pp 154–165
8. Temuçin YH, Sojoodi A, Alizadeh P, Afsahi A (2021) Efficient multi-path NVLink/PCIe-aware UCX based collective communication for deep learning. In: 2021 IEEE Symposium on High-Performance Interconnects (HOTI). IEEE, pp 25–34
9. Alizadeh P, Sojoodi A, Hassan Temucin Y, Afsahi A (2022) Efficient process arrival pattern aware collective communication for deep learning. In: Proceedings of the 29th European MPI Users' Group Meeting, pp 68–78
10. NVLink and NVSwitch. https://www.nvidia.com/en-us/data-center/nvlink/. Accessed 2023-06-30
11. Pentakalos OI (2002) An introduction to the Infini-Band architecture. In: International CMG Conference 2002, Reno, USA, pp 425–432
12. Kim J, Dally WJ, Scott S, Abts D (2008) Technology-driven, highly-scalable Dragonfly topology. In: 2008 International Symposium on Computer Architecture. IEEE, pp 77–88
13. Camara JM, Moreto M, Vallejo E, Beivide R, Miguel-Alonso J, Martínez C, Navaridas J (2010) Twisted torus topologies for enhanced interconnection networks. IEEE Trans Parallel Distrib Syst 21(12):1765–1778
14. Jain N, Bhatele A, Howell LH, Böhme D, Karlin I, León EA, Mubarak M, Wolfe N, Gamblin T, Leininger ML (2017) Predicting the performance impact of different Fat-Tree configurations. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp 1–13
15. Chunduri S, Harms K, Parker S, Morozov V, Oshin S, Cherukuri N, Kumaran K (2017) 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, pp 1–13
16. Yu H, Chung I-H, Moreira J (2006) Topology mapping for Blue Gene/L supercomputer. In: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, p 116
17. Jyothi SA, Singla A, Godfrey PB, Kolla A (2016) Measuring and understanding throughput of network topologies. In: SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, pp 761–772
18. Top500, MARCONI-100. https://www.top500.org/system/179845/. Accessed 2023-07-01
19. Shpiner A, Haramaty Z, Eliad S, Zdornov V, Gafni B, Zahavi E (2017) 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, pp 1–8
20. Zhou Z, Yang X, Lan Z, Rich P, Tang W, Morozov V, Desai N (2015) Improving batch scheduling on blue Gene/Q by relaxing 5d torus network allocation constraints. In: 2015 IEEE International Parallel and Distributed Processing Symposium. IEEE, pp 439–448
21. Tang W, Desai N, Buettner D, Lan Z (2010) 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, pp 1–11
22. Skinner D, Kramer W (2005) Understanding the causes of performance variability in HPC workloads. In: IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005. IEEE, pp 137–149
23. Afzal A, Hager G, Wellein G (2023) The role of idle waves, desynchronization, and bottleneck evasion in the performance of parallel programs. IEEE Transa Parallel Distrib Syst 34(02):623–638
24. Bhatele A, Thiagarajan JJ, Groves T, Anirudh R, Smith SA, Cook B, Lowenthal DK (2020) The case of performance variability on Dragonfly-based systems. In: 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, pp 896–905
25. Chester D, Groves T, Hammond SD, Law TR, Wright SA, Smedley-Stevenson R, Fahmy SA, Mudalige GR, Jarvis S (2021) Stressbench: a configurable full system network and I/O benchmark framework. In: IEEE High Performance Extreme Computing Conference. York

26. Propagation and Decay of Injected One-Off Delays on Clusters: A Case Study | IEEE Conference PublicationIEEE Xplore. https://ieeexplore.ieee.org/document/8890995. Accessed 02/04/2024
27. Salimi Beni M, Crisci L, Cosenza B (2023) EMPI: enhanced message passing interface in modern c++. In: 2023 23rd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid). IEEE
28. Wang X, Mubarak M, Yang X, Ross RB, Lan Z (2018) 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, pp 1113–1122
29. De Sensi D, Di Girolamo S, Hoefler T (2019) Mitigating network noise on Dragonfly networks through application-aware routing. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp 1–32
30. Liu Y, Liu Z, Kettimuthu R, Rao N, Chen Z, Foster I (2019) Data transfer between scientific facilities - bottleneck analysis, insights and optimizations. In: 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp 122–131
31. Kousha P, Sankarapandian Dayala Ganesh Ram KR, Kedia M, Subramoni H, Jain A, Shafi A, Panda D, Dockendorf T, Na H, Tomko K (2021) Inam: cross-stack profiling and analysis of communication in MPI-based applications. In: Practice and Experience in Advanced Research Computing, pp 1–11
32. Brown KA, McGlohon N, Chunduri S, Borch E, Ross RB, Carothers CD, Harms K (2021) A tunable implementation of Quality-of-Service classes for HPC networks. In: International Conference on High Performance Computing. Springer, pp 137–156
33. Suresh KK, Ramesh B, Ghazimirsaeed SM, Bayatpour M, Hashmi J, Panda DK (2020) Performance characterization of network mechanisms for non-contiguous data transfers in MPI. In: 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE, pp 896–905
34. Hemmert KS, Bair R, Bhatale A, Groves T, Jain N, Lewis C, Mubarak M, Pakin SD, Ross R, Wilke JJ (2020) Evaluating trade-offs in potential exascale interconnect technologies. Lawrence Livermore National Lab. (LLNL), Livermore
35. Cheng Q, Huang Y, Bahadori M, Glick M, Rumley S, Bergman K (2018) Advanced routing strategy with highly-efficient fabric-wide characterization for optical integrated switches. In: 2018 20th International Conference on Transparent Optical Networks (ICTON). IEEE, pp 1–4
36. Zacarias FV, Nishtala R, Carpenter P (2020) Contention-aware application performance prediction for disaggregated memory systems. In: Proceedings of the 17th ACM International Conference on Computing Frontiers, pp 49–59
37. Ponce M, Zon R, Northrup S, Gruner D, Chen J, Ertinaz F, Fedoseev A, Groer L, Mao F, Mundim BC et al (2019) 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), pp 1–8
38. Marconi100. The new accelerated system. https://www.hpc.cineca.it/hardware/marconi100. Accessed 2023-07-01
39. Kang Y, Wang X, McGlohon N, Mubarak M, Chunduri S, Lan Z (2019) Modeling and analysis of application interference on Dragonfly+. In: Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, pp 161–172
40. Wang X, Mubarak M, Kang Y, Ross RB, Lan, Z (2020) Union: an automatic workload manager for accelerating network simulation. In: 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp 821–830
41. Salimi Beni M, Cosenza B (2023) An analysis of long-tailed network latency distribution and background traffic on dragonfly+. In: International Symposium on Benchmarking, Measuring and Optimization. Springer, pp 123–142
42. Beni MS, Cosenza B (2022) An analysis of performance variability on Dragonfly+ topology. In: 2022 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, pp 500–501
43. Navaridas J, Lant J, Pascual JA, Lujan M, Goodacre J (2019) Design exploration of multi-tier interconnection networks for exascale systems. In: Proceedings of the 48th International Conference on Parallel Processing, pp 1–10
44. Hashmi JM, Xu S, Ramesh B, Bayatpour M, Subramoni H, Panda DKD (2020) Machine-agnostic and communication-aware designs for MPI on emerging architectures. In: 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, pp 32–41

45. Subramoni H, Lu X, Panda DK (2017) A scalable network-based performance analysis tool for MPI on large-scale HPC systems. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, pp 354–358
46. Teh MY, Wilke JJ, Bergman K, Rumley S (2017) Design space exploration of the Dragonfly topology. In: International Conference on High Performance Computing. Springer, pp 57–74
47. Zahn F, Fröning H (2020) On network locality in MPI-based HPC applications. In: 49th International Conference on Parallel Processing-ICPP, pp 1–10
48. Hoefler T, Schneider T, Lumsdaine A (2010)Characterizing the influence of system noise on large-scale applications by simulation. In: SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, pp 1–11
49. Maricq A, Duplyakin D, Jimenez I, Maltzahn C, Stutsman R, Ricci R (2018) Taming performance variability. In: 13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18), pp 409–425
50. Vetter J, Chambreau C (2005) MPIP: lightweight, scalable MPI profiling
51. Arnold DC, Ahn DH, De Supinski B, Lee G, Miller B, Schulz M (2007) Stack trace analysis for large scale applications. In: 21st IEEE International Parallel & Distributed Processing Symposium (IPDPS'07), Long Beach, CA
52. Petrini F, Kerbyson DJ, Pakin S (2003) The case of the missing supercomputer performance: achieving optimal performance on the 8,192 processors of asci q. In: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, p 55
53. Sato K, Ahn DH, Laguna I, Lee GL, Schulz M, Chambreau CM (2017) Noise injection techniques to expose subtle and unintended message races. In: Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp 89–101
54. Smith SA, Cromey CE, Lowenthal DK, Domke J, Jain N, Thiagarajan JJ, Bhatele A(2018) Mitigating inter-job interference using adaptive flow-aware routing. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, pp 346–360
55. McGlohon N, Carothers CD, Hemmert KS, Levenhagen M, Brown KA, Chunduri S, Ross RB (2021) 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, pp 40–50
56. Shah A, Müller M, Wolf F (2018) Estimating the impact of external interference on application performance. In: Euro-Par 2018: Parallel Processing: 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27-31, 2018, Proceedings 24. Springer, pp 46–58
57. Zhang Y, Groves T, Cook B, Wright NJ, Coskun AK (2020)Quantifying the impact of network congestion on application performance and network metrics. In: 2020 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, pp 162–168
58. Brown KA, Jain N, Matsuoka S, Schulz M, Bhatele A (2018) Interference between I/O and MPI traffic on Fat-Tree networks. In: Proceedings of the 47th International Conference on Parallel Processing, pp 1–10
59. Tang X, Zhai J, Qian X, He B, Xue W, Chen W (2018) Vsensor: leveraging fixed-workload snippets of programs for performance variance detection. In: Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp 124–136
60. Zheng L, Zhai J, Tang X, Wang H, Yu T, Jin Y, Song SL, Chen W (2022) Vapro: performance variance detection and diagnosis for production-run parallel applications. In: Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp 150–162
61. Besta M, Schneider M, Konieczny M, Cynk K, Henriksson E, Di Girolamo S, Singla A, Hoefler T (2020) 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, pp 1–18
62. Kang Y, Wang X, Lan Z (2020) 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, pp 189–200
63. Newaz MN, Mollah MA, Faizian P, Tong Z (2021) Improving adaptive routing performance on large scale Megafly topology. In: 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid). IEEE, pp 406–416
64. Mollah MA, Wang W, Faizian P, Rahman MS, Yuan X, Pakin S, Lang M (2019) Modeling universal globally adaptive load-balanced routing. ACM Trans Parallel Comput 6(2):1–23

65. Faizian P, Alfaro JF, Rahman MS, Mollah MA, Yuan X, Pakin S, Lang M (2018) Tpr: traffic pattern-based adaptive routing for Dragonfly networks. IEEE Trans Multi-Scale Comput Syst 4(4):931–943
66. De Sensi D, Di Girolamo S, McMahon KH, Roweth D, Hoefler T (2020) An in-depth analysis of the slingshot interconnect. In: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, pp 1–14
67. Wen K, Samadi P, Rumley S, Chen CP, Shen Y, Bahadori M, Bergman K, Wilke J (2016)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, pp 166–177
68. Rahman MS, Bhowmik S, Ryasnianskiy Y, Yuan X, Lang M (2019) Topology-custom ugal routing on Dragonfly. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC '19. Association for Computing Machinery, New York, NY, USA
69. Rocher-Gonzalez J, Escudero-Sahuquillo J, Garcia PJ, Quiles FJ, Mora G (2019) 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, pp 221–230
70. Kaplan F, Tuncer O, Leung VJ, Hemmert SK, Coskun AK (2017) 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, pp 325–334
71. Michelogiannakis G, Ibrahim KZ, Shalf J, Wilke JJ, Knight S, Kenny JP (2017) 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, pp 228–237
72. Zhang Y, Tuncer O, Kaplan F, Olcoz K, Leung VJ, Coskun AK (2018) Level-spread: a new job allocation policy for Dragonfly networks. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, pp 1123–1132
73. Wang X, Yang X, Mubarak M, Ross RB, Lan Z (2017) A preliminary study of intra-application interference on Dragonfly network. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, pp 643–644
74. Aseeri SA, Gopal Chatterjee A, Verma MK, Keyes DE (2021) A scheduling policy to save 10% of communication time in parallel fast Fourier transform. Concurr Comput Pract Exp 35:e6508
75. Glass CJ, Ni LM (1992) The turn model for adaptive routing. ACM SIGARCH Comput Architect News 20(2):278–287
76. OSU Micro-Benchmarks 5.8, The Ohio State University.https://mvapich.cse.ohio-state.edu/benchmarks/. Accessed 2023-07-01
77. Heitmann K, Finkel H, Pope A, Morozov V, Frontiere N, Habib S, Rangel E, Uram T, Korytov D, Child H et al (2019) The outer rim simulation: a path to many-core supercomputers. Astrophys J Suppl Ser 245(1):16
78. Heroux MA, Doerfler DW, Crozier PS, Willenbring JM, Edwards HC, Williams A, Rajan M, Keiter ER, Thornquist HK, Numrich RW (2009) Improving performance via mini-applications. Sandia National Laboratories, Technical Report SAND2009-5574, vol 3
79. Hunold S, Carpen-Amarie A (2016) Reproducible MPI benchmarking is still not as easy as you think. IEEE Trans Parallel Distrib Syst 27(12):3617–3630
80. Slurm, Slurm's job allocation policy for Dragonfly network. https://github.com/SchedMD/slurm/blob/master/src/plugins/select/linear/select_linear.c. Accessed 2023-07-01
81. GitHub - cea-hpc/hp2p: Heavy Peer To Peer: a MPI based benchmark for network diagnostic. https://github.com/cea-hpc/hp2p. Accessed 15 May 2023
82. (2008) Pearson's correlation coefficient. In: Kirch W (eds) Encyclopedia of public health. Springer, Dordrecht. https://doi.org/10.1007/978-1-4020-5614-7_256
83. Zar JH (2005) Spearman rank correlation. In: Encyclopedia of biostatistics, vol 7. https://doi.org/10.1002/0470011815.b2a15150
84. Hunold S, Carpen-Amarie A (2018) Autotuning MPI collectives using performance guidelines. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, pp 64–74
85. Hunold S, Carpen-Amarie A (2018) Algorithm selection of MPI collectives using machine learning techniques. In: 2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS). IEEE, pp 45–50

86. Hunold S, Steiner S (2022) OMPICollTune: Autotuning MPI collectives by incremental online learning. In: 2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS). IEEE, pp 123–128
87. Salimi Beni M, Hunold S, Cosenza B (2023) Algorithm selection of MPI collectives considering system utilization. In: Euro-Par 2023: Parallel Processing Workshops. Springer
88. Dean J, Barroso LA (2013) The tail at scale. Commun ACM 56(2):74–80. https://doi.org/10.1145/2408776.2408794