FORS: Fault-adaptive Optimized Routing and Scheduling for DAQ Networks

Eloise Stein[1,2*], Quentin Bramas[1], Flavio Pisani[2], Tommaso Colombo[2], Cristel Pelsser[1,3]

[1*]University of Strasbourg, Strasbourg, State, France.
[2]European Council for Nuclear Research (CERN), Geneva, Switzerland.
[3]UCLouvain, Ottignies-Louvain-la-Neuve, Belgium.

*Corresponding author(s). E-mail(s): eloise.stein@etu.unistra.fr;
Contributing authors: bramas@unistra.fr; flavio.pisani@cern.ch;
tommaso.colombo@cern.ch; cristel.pelsser@uclouvain.be;

Abstract

Data acquisition (DAQ) networks, widely used in scientific research and industrial applications, are composed of numerous interconnected servers, exchanging substantial data volumes produced by large scientific instruments. One traffic matrix generally used in such networks is the all-to-all collective exchange, which demands substantial network resources, making network failures particularly challenging to mitigate. If not mitigated, the effects of network failures severely hamper the performance of the DAQ network, potentially leading to the loss of valuable experimental data.

In the context of DAQ networks using a fat-tree topology, we propose FORS: a scheduling and associated routing solution to support the all-to-all collective exchange under network failures. FORS optimizes bandwidth utilization in the face of any failure scenarios, ensuring robust performance compared to the existing approaches. We propose an algorithm to solve the scheduling. For the routing, we design an algorithm for simple failure scenarios, along with a linear programming model to address more complex failure scenarios. We validate our proposed solution using a real-world DAQ network as a case study. Results demonstrate significant performance degradation in existing approaches and FORS' consistent ability to achieve higher throughput across various failure scenarios.

Keywords: all-to-all, fat-tree networks, integer linear programming, optimal routing, fault-tolerance, data acquisition

1 Introduction

Typically composed of a diverse set of sensors, DAQ systems capture large amounts of data. These systems find extensive use in various fields, including scientific research such as aerospace [1–3], healthcare [4, 5] and physics [6–8]. For instance, DAQ systems at the European Organization for Nuclear Research (CERN)[7, 8] process tens of exabytes annually[9, 10], significantly contributing to advancements in the field of physics research[11].

Data acquisition (DAQ) systems often rely on HPC applications for real-time analysis and efficient processing of substantial volumes of data. Optimization of HPC applications is the subject of much research, pushing back the limits of components[12, 13], architecture[14–16] and networks[17]. A lack of optimization can result in a significant loss of performance, preventing important scientific discoveries from being made. While the optimization of collective exchanges in HPC applications is extensively discussed and addressed [18–25], to the best of our knowledge, there has been no proposals on optimizing these collective exchanges in the event of network failures, involving bandwidth reduction, despite the fact that failures are common [26]. As we demonstrate in this paper, network performance is currently significantly impacted when bandwidth reduction occurs. Addressing this issue is challenging as it involves adapting these collective exchange scheduling and routing to the remaining network bandwidth during failures.

Here, we focus on networks built to execute all-to-all exchanges **all the time**, such as DAQ networks needed by high-energy physics experiments. The data exchanged is produced in real-time by large scientific instruments. For instance, the Large Hadron Collider (LHC) at CERN accelerates particles to energies of up to 6.8 TeV and makes them collide to energies up to 13.6 TeV. Numerous sensors capture various aspects of the resulting collisions, the events. The sensors are connected to servers to which they send their data. Then, each server needs to exchange the data produced by its sensors with the other servers in an all-to-all collective exchange, to reconstruct the collisions. The all-to-all collective exchange demands significant network resources and is highly sensitive to failures. Furthermore, since data is produced by sensors in real-time, servers must reconstruct the events promptly. A delay in reconstruction can overload server buffers with data, leading to congestion and the potential loss of important data. Consequently, throughput is the focal network metric in this paper.

Collective operations, in general, are also increasingly used in machine learning[27–30], including AllGather, AllReduce, or ReduceScatter [31]. However, they differ from the all-to-all collective exchange. Our problem is specific to DAQ systems.

In this paper, we propose a Fault-adaptive Optimized Routing and Scheduling (FORS) solution to maintain high all-to-all throughput despite the bottlenecks introduced by network link failures. Our first contribution measures on the real DAQ the reduced throughput in various failure scenarios (Section 3). Subsequently, we introduce in Section 4 the first element of FORS, which is an algorithm to adapt the scheduling of the communications for the all-to-all collective exchange in case of link failures. Additionally, FORS is composed of a semi-algorithmic routing solution, presented in Section 5, combining a route computation algorithm for basic failure scenarios with

an Integer Linear Programming (ILP) model designed to address more specific combination of failures. The purpose of these two algorithms is to provide congestion-free paths between servers within the network based on the given failure scenario. We demonstrate in Section 6 the applicability and performance of our solution on a real, operational DAQ network at CERN employing HPC for processing large volumes of scientific data. We compare our proposal with currently deployed approaches. The structure and operation of this network, detailed in Section 2, relies on a fat-tree topology with all-to-all exchange.

2 Background

In this section, we introduce the fat-tree topology and all-to-all collective exchange common to DAQ networks. Subsequently, we focus on our case of study and describe its specifics and existing routing algorithms.

2.1 The Fat-tree topology

Our solution currently focuses on a single topology family: Generalized Fat-trees, more specifically k-ary-L-trees [32]. Figure 2 pictures an example of such a topology. Fat-tree topologies are widely used in HPC. Many of the systems on the latest Top500 list employ a fat-tree topology [33]. Generalized fat-tree topologies are particularly well-suited for the all-to-all collective exchange, thanks to their scalability and rearrangeable non-blocking nature[34]. This last characteristic allows each end node to communicate at the same time, making the best use of the available bandwidth [35].

The all-to-all collective exchange has been extensively studied and applied in various widely-used topologies within HPC, such as the mesh [36] and torus [37] [38] topologies. These topologies are still used and in place, but the most popular ones are currently Dragonfly [39] and Fat-tree for their scalability.

Dragonfly [19] and Dragonfly+ [40, 41] topologies are suitable for many HPC workloads, but not the best for the all-to-all collective exchange. Dragonfly leverages the locality of data exchanges observed in some HPC applications, but not in all-to-all. They are blocking topologies except intra-group, which makes them less suitable for all-to-all exchanges. Furthermore, the completion time of all-to-all exchanges with dragonfly topologies is increased compared to running the application on fat-tree topologies.

2.2 The all-to-all collective exchange

The all-to-all collective exchange is a communication pattern in which each process (or server) sends data to every other process and receives data from every other process in the group. This communication pattern finds common application in parallel or distributed computing environments and is standardized in Message Passing Interface (MPI) [42]. This data exchange plays a crucial role in various scientific applications, such as the Fast Fourier Transform (FFT) algorithm [38, 43, 44], which finds application in diverse scientific fields, including healthcare with medical image reconstruction in Magnetic Resonance Imaging (MRI) [45]. Moreover, the all-to-all exchange is widely

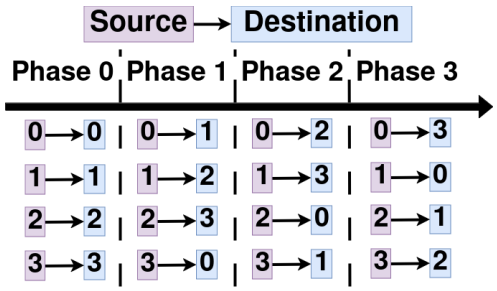

Fig. 1: The Linear-shift scheduling with 4 processes. Each process communicates with all other processes. To prevent congestion, the exchange is divided into 4 phases. In each phase, each process sends data to another process and receives data from another one. The minimum number of phases for the all-to-all exchange is the same as the number of processes.

used in Grid-based Simulations, employed in the prediction of weather patterns [46] or in Hadoop-like applications [47].

Simultaneous all-to-all data transfer can cause network congestion. In a network connecting a group of servers denoted as S, each hosting a single process, each server must send and receive data to and from every other server. A full-mesh topology, where each server has a direct link to all others, avoids congestion but is a waste of resources as it requires a bandwidth of $|S|^2$ when $|S|$ bandwidth units are sufficient at the servers. To reach this lower bandwidth, a commonly employed strategy is to divide the time into synchronized phases. Each server communicates with exactly one server in each phase. After all phases, every server has exchanged data with all others, needing only $|S|$ bandwidth units per phase.

To complete an all-to-all exchange with synchronized phases, communications are scheduled using an algorithm that computes the unique destination for each source and phase. An example of such an algorithm is the **Linear-shift** pattern [22] illustrated in Figure 1, where each destination process is given by $d = (s + p) \mod S$, where s refers to the source process, p is the phase and S is the total number of processes. Many other scheduling algorithms [24] exist, such as XOR or Bandwidth-optimal [18]. As network links are used at full capacity by communication flows, combining these algorithms with optimal routing helps avoid congestion in the network and ensure that no communication flows share links.

A persistent issue remains: in the event of network failures, existing scheduling algorithms do not take into account the loss of bandwidth. In such scenarios, multiple communication flows end up sharing links on the network, resulting in congestion and decreased throughput, as we show in Section 3. One of our contributions is a novel scheduling algorithm, described in Section 4, which considers bandwidth reduction and offers a conflict-free scheduling for all failure scenarios.

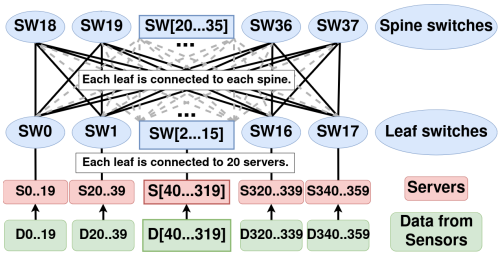

Fig. 2: The fat-tree topology of the studied DAQ Network. The notation of the fat-tree topology is FT($L = 2$;$M_0 = 20$,$M_1 = 18$), where L defines the number of layers, M_0 is the number of spine switches and M_1 is the number of leaf switches. The number of servers is $M_0 * M_1 = 360$.

2.3 Case of study: A DAQ network

The application of the all-to-all collective exchange with a fat-tree topology is used in various systems [46], including data acquisition (DAQ) in particle physics experiments [48–50] at the European Council for Nuclear Research (CERN). Our case study is the DAQ network of the LHCb experiment at CERN depicted in Figure 2.

The Large Hadron Collider (LHC) at CERN stands as the world's largest and most powerful particle accelerator. Within this accelerator, two particle beams are accelerated to nearly the speed of light before colliding. The resulting physical phenomena from these collisions are studied by physicists, significantly advancing the field of physics research [51]. Throughout the remainder of this paper, a physical phenomenon is referred to as **an event**. After over three years of upgrades and maintenance, the LHC currently achieves a record energy level of 13.6 TeV. To address such high energy levels, the DAQ system of the LHCb experiment has also experienced major upgrades. The DAQ network can now achieve a throughput of up to 46 Tbps, surpassing the design target of 32 Tb/s[52]. Such performance makes the studied network the one with the highest real-time data bandwidth compared to other experiments at CERN [53, 54].

Instruments responsible for measuring the properties of these events are known as detectors, typically composed of multiple sensors. The data produced by the sensors represent an image of events resulting from collisions between the beams at a given time. One of the objectives of a DAQ system is to reconstruct a comprehensive view of each event generated in the LHC. This process is called **Event Building** and is generally performed over a high-throughput network of servers. The sensors are directly connected to the servers and send data to them. The servers then perform an all-to-all collective exchange to reconstruct the events generated in the LHC. The scheduling pattern of the communication between servers in our studied network is computed using the Linear-shift[22] algorithm introduced in Section 2.2.

The studied DAQ network relies on the Infiniband technology. Infiniband is widely used in HPC[33]. The fat-tree topology of the studied DAQ network, illustrated in Figure 2, is composed of 28 40-ports Infiniband High Data Rate (HDR) switches. Throughout this paper, the switches located at the top layer of the fat-tree topology

are referred to as **spine switches** while those at the bottom layer are denoted as **leaf switches**. The studied fat-tree topology includes 18 leaf switches and 20 spine switches. Each leaf switch is connected to every spine with a 200 Gbps optical fiber. Leaves are also directly connected to at most 20 servers with 20 copper cables with the same capacity of 200 Gbps. The network enables the connection of a total of 360 servers. However, the effective number of servers utilized in the network is 326. The total throughput that can theoretically be achieved by the DAQ application is $326 * 200 = 65200$ Gbps $= 65.2$ Tbps. Nevertheless, due to various overheads, our experiments show in Section 6 that the actual total throughput achieved by the DAQ application is approximately 46 Tbps. This represents a utilization of about 70% of the available bandwidth in the network.

Every Infiniband network relies on a controller, called OpenSM [55]. The controller is responsible for computing routes and pushing them into the Linear-Forwarding Tables (LFT) of the switches [55]. To efficiently route each communication flow, a routing algorithm compatible with the linear-shift [22] scheduling is used to avoid congestion in the topology. Ensuring high throughput requires selecting the shortest paths for each communication flow. In our studied network, every inter-leaf flow must traverse the topology from the leaf switches to the spine switches and then back to the leaf switches, this path being the shortest in the topology of the DAQ network. Hence, the challenge lies in choosing a spine for each communication flow. To prevent two flows from using the same link simultaneously, each leaf must employ a distinct spine for each communication flow. A routing algorithm with those characteristics is provided by Infiniband. This routing algorithm, named Ftree [55], uniformly distributes the traffic on the links between spine and leaf switches according to the destination server. Consequently, the linear-shift algorithm combined with the Ftree routing algorithm ensures that there is no congestion in the network when there are no network failures [22].

In the event of link failures, the use of the Ftree routing is not recommended as the topology is no longer a pure fat-tree. In such scenarios, the default behavior of OpenSM is to switch from Ftree to Min-Hop, upon the detection of failures. Similar to Ftree, Min-Hop computes the shortest path for each communication flow and uniformly distributes the traffic. Contrary to Ftree, Min-hop is applicable on non fat-tree topologies. In the presence of a link failure, there may not be sufficient links available to allocate traffic without congestion. Consequently, the same spine might need to be used simultaneously by multiple communication flows from and to a leaf switch. Min-Hop prioritizes the spine with the least communication flows to balance the traffic uniformly [55].

In the next section, we show the performance of the studied DAQ application with the routing and scheduling algorithms currently in use, alongside viable alternatives. To motivate our problem, we demonstrate how performance degrades significantly in case of failures.

3 Motivation

The objective of this section is to demonstrate the insufficient performance of the combined linear shift scheduling and Infiniband routing algorithms in the event of network failures. Network failures are common in data centers [26]. In [56], the authors show a comprehensive study of network failures in data centers which presents statistical findings based on a year-long measurement of failures in a real data center network. Despite the generally reliable nature of data center networks, failures occurred on a daily basis during the measurement period. Additionally, the study reveals that some link failures can last for an extended period due to repair difficulties, either because of a lack of spare network equipment or the physical access of the link being less reachable.

Furthermore, there is a time correlation between failures in large-scale distributed systems. In [57], the authors highlight the existence of peak periods during which failures are most pronounced.

The literature shows that failures in large-scale networks are frequent, time-correlated, and can last a long time. In this section, we evaluate the consequences of failures with various Infiniband routing algorithms for all-to-all scheduling with synchronization. In our production DAQ, which is not capturing usable physics data during the LHC warmup phase, we demonstrate that even a few network failures result in a significant loss of throughput. Such failures in the LHC full operation phase would lead to a significant loss of valuable data.

3.1 Experiments setup

To illustrate the performance degradation during bandwidth reduction, We count the number of events reconstructed by each server over 5-second intervals in the all-to-all application. The duration of the measurement is one minute for each failure scenario, resulting in a total of 12 data points. Multiplying this rate by the volume of data of an event, we obtain the global throughput of the DAQ network.

We perform our measurements under various failure scenarios, involving links between leaf and spine switches. The links between the leaf switches and the servers are not considered, since failures on those links disconnect servers and, hence sources, from the network. In such cases, optimizing the bandwidth is not meaningful as we cannot reconstruct events anyway. To conduct our measurements, we manually disable links between leaf and spine switches. In the following description of the failures, the IDs of the switches are referenced according to Figure 2:

- 0F: No failures.
- xF SW_0: $x \in [1, 2, 3]$ failure(s) between the leaf switch SW_0 and random spine switches. This scenario allows to show the impact of bandwidth reduction located on a single leaf switch.
- xF $SW_{0,5,11}$: $x \in [1, 2, 3]$ failure(s) between the leaf switches SW_0, SW_5 and SW_{11} and consecutive spine switches. All cited leaf switches have failures with a different spine. For instance, 1F $SW_{0,5,11}$ corresponds to the link failures SW_0-SW_{18}, SW_5-SW_{19} and SW_{11}-SW_{20}. This scenario illustrates the impact of multiple link failures occurring in more diverse locations inside the network.

- $SW_{18,19}$: The spine switches SW_{18} and SW_{19} have failed. This scenario shows the impact of spine failures.
- $SW_{18,19,22,23}$: The spine switches SW_{18}, SW_{19}, SW_{22} and SW_{23} have failed.

We consider a large variety of realistic[57, 58] failure scenarios. The literature [26] shows that grouped failures rarely consist of more than five failures simultaneously. Our failure scenarios range from 1 failure (XF SW_0) to 9 failures (3F $SW_{0,5,11}$) occurring simultaneously. Additionally, we evaluate the performance of the different approaches with spine failures, ranging from 40 to 80 link failures in the network. While these latter failure scenarios are rather extreme our results show that the performance degradation is identical to some more frequent lighter failure cases.

As explained in Section 2.2, to prevent congestion, the all-to-all collective exchange is segmented into phases. The DAQ application running on the servers ensures that all sources finish sending their data to the current destinations before moving to the next phase. The synchronization is performed using the Tree-based Barrier algorithm [59]. This approach has demonstrated better performance than unsynchronized communications without failures, especially in large-scale systems [52].

The routing algorithm used in the absence of failures is Ftree, as explained in Section 2, which is the optimal routing algorithm for the linear-shift scheduling [22]. The routing algorithm Ftree is employed in scenario 0F when no bandwidth reduction occurred. It is also used in the scenarios $SW_{18,19}$ and $SW_{18,19,22,23}$, denoting the failure of the spine switches SW_{18}, SW_{19}, SW_{22} and SW_{23}. Despite the complete failure of those switches, the Infiniband controller does not fail over to Min-Hop as the topology stays a fat-tree in such cases. In the other failure scenarios, the controller switches to the Min-Hop routing algorithm, which is the default behavior in the absence of a fat-tree, and which is recommended as it balances the traffic better than Ftree in case of bandwidth reduction [60]. Other OpenSM routing algorithms exist, such as Up-Down, LAyered SHortest Path Routing (LASH), and Dimension Order Routing (DOR), but they are not suitable for fat-tree topologies [55]. [61] studies the performance of these algorithms and shows that they are all highly sensitive to network failures.

Here we evaluate the performance of Infiniband routing algorithms optimized for fat-tree topologies (Ftree and Min-Hop), in both normal operation and during bandwidth reduction, on a real Infiniband network. Additionally, we incorporate the adaptive routing version of Ftree (Ftree-AR) into our measurements. This adaptive routing version of Ftree is already implemented in Infiniband [55] but is not normally used by the DAQ network under study. This approach dynamically adjusts routing in the event of failures by setting multiple alternative routes in the routing tables and balancing the traffic on the paths depending on the load.

3.2 Measured throughput

In the synchronized all-to-all exchange, link failures affect all servers. Link failures reduce the usable bandwidth for all servers. Our results show the same throughput with two failures on a single leaf switch ($2FSW_0$) and with the loss of two spines ($SW_{18,19}$). In the first case, we loose two paths for the servers below the leaf switch. In the latter, all leaf switches loose two paths for their servers. Because the

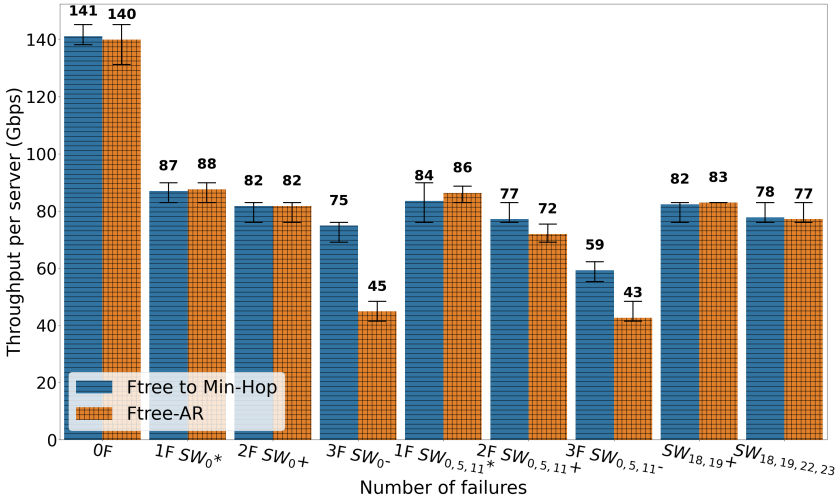

Fig. 3: The average throughput achieved per server according to the failure scenario with the default routing algorithms Ftree/Min-Hop and the adaptive routing version of Ftree. To obtain the total throughput achieved by the DAQ application, one can multiply this value by the number of servers in the DAQ application, which is 326. The error bars represent the minimum, median and maximum values. The topology of the data acquisition network is illustrated in Figure 2. Failure scenarios with the '*' symbol correspond to a bandwidth reduction of 1, those with the '+' symbol to a reduction of 2 and '-' corresponds to a reduction of 3.

exchange is synchronized, the servers need to wait on each other. As a result, the communications for all leaves are slowed down similarly in both cases. We have a uniform bandwidth reduction for all servers. Likewise the loss of two links on one leaf switch and the loss of two links on all leaf switches lead to the same bandwidth reduction. Furthermore, the throughput with the failure of two spine switches (scenario $SW_{18,19}$ in Figure 3), being the failure of 40 links, is better than the throughput with three failures on a single leaf switch (scenario 3F SW_0 in Figure 3). Losing two spines results in a bandwidth reduction of 2 paths per leaf switch, as all leaf switches lose 2 links, while the loss of three links on a single leaf switch results in a bandwidth reduction of 3 paths for all leaf switches. Therefore, the latter scenario degrades the throughput more than the loss of two spines as the exchange is synchronized.

The location of failures has an influence on the throughput in the synchronized all-to-all exchange. It is interesting to note that scenarios where failures are not located on the same spine switch exhibit lower throughput. Again, this is because more paths become unavailable between the leaves. For instance, scenario $SW_{18,19}$, representing the failure of the spines SW_{18} and SW_{19}, which is the loss of 40 links, demonstrates higher throughput than scenario 3F $SW_{0,5,11}$, where three leaf switches experience three link failures with different spines, which is the loss of 9

links. This behavior becomes more pronounced as the number of failures increases. In contrary, the bandwidth reduction for synchronized exchange is expected to be uniform across all servers, explaining why scenario 2F SW_0 and $SW_{18,19}$ have the same throughput, while 2F SW_0 is the loss of 2 links and $SW_{18,19}$ is the loss of 40 links (as explained in the previous paragraph).

Coming back to scenarios 3F $SW_{0,5,11}$ and $SW_{18,19}$, the difference in throughput arises from the fact that in the scenarios 3F $SW_{0,5,11}$, the number of paths between the leaf switches impacted by failures is higher compared to scenario $SW_{18,19}$. In the former scenario, SW_0 experiences one link failure with $SW_{18,19,20}$, SW_5 with $SW_{21,22,23}$ and SW_{11} with $SW_{24,25,26}$, which makes the number of possible paths between SW_0, SW_5 and SW_{11} reduced to 11, as there are only 11 spines where the leaves $SW_{0,5,11}$ are all connected. This results in a bandwidth reduction of 9, as there are 20 spines in the topology. Consequently, the scenario 3F $SW_{0,5,11}$ further reduces the bandwidth between the leaf switches impacted by failures, resulting in increased congestion and decreased throughput compared to the loss of 2 spine switches. Appendix B provides more details on the complexity of this failure scenario.

The performance of Ftree adaptive routing is reduced or equivalent compared to the performance of Min-Hop and Ftree. The use of adaptive routing demonstrates performance equivalent or inferior to the Ftree and Min-Hop routing solutions. Note that Ftree-AR behaves exactly the same as Ftree without any failures. The throughput of Ftree and Min-Hop is equivalent to Ftree adaptive routing with synchronized all-to-all, except in the scenarios 3F SW_0 and $3F$ $SW_{0,5,11}$, in which the throughput is lower. This is attributed to the fact that the traffic generated by the DAQ application is too bursty, as datacenter traffic in general [62] [63] [47], for the Ftree adaptive routing algorithm to adapt accordingly. Beyond our measurements, another source has indicated that adaptive routing leads to more performance degradation for bursty traffic compared to deterministic routing [64].

In all approaches, it becomes evident that even a few failures on a network comprising 360 links lead to a notable degradation in the performance of the DAQ network. As described in Section 2, the current application employs synchronized all-to-all exchange with the Ftree routing algorithm under normal conditions, while switching to Min-Hop routing in the event of failures. As illustrated in Figure 3, the throughput per server decreases from 141 Gbps to 87 Gbps with a single link failure. Multiplying these values by the number of servers, 326, connected in the DAQ network, we obtain the total network throughput. Consequently, the total throughput of the DAQ network decreases from 45.9 Tbps to 28.3 Tbps with a single link failure. This provides evidence that the current scheduling and routing (Ftree to Min-Hop and the adaptive routing version of Ftree) combinations are not sufficient. Finding a more adaptive scheduling and routing solution in the event of bandwidth reduction is crucial.

4 Fault-adaptive scheduling algorithm

In this section, we present our new scheduling algorithm performing a congestion-free all-to-all exchange in an arbitrary 2-layers fat-tree. The challenge is using the

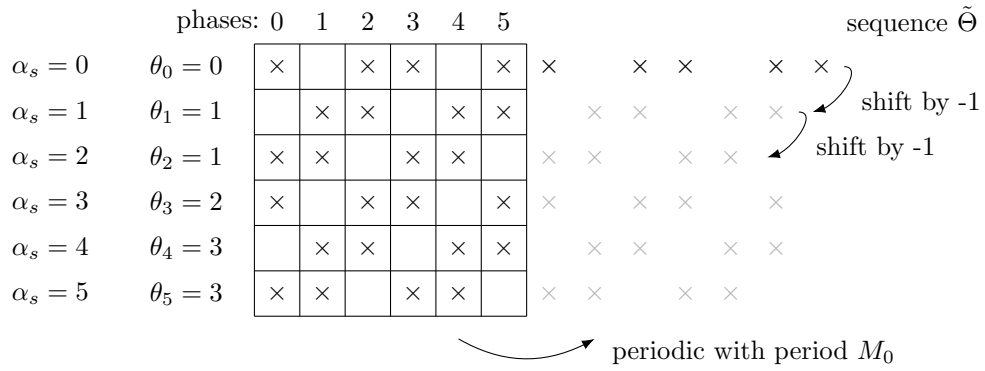

Fig. 4: Illustration of function $T(s, p)$, with $M_0 = 6$ and $f = 2$, that defines whether a server s transmits at phase p or not. The function depends only on α_s and is periodic of period M_0 for the second variable. Server s transmits at phase p if and only if the corresponding cell is crossed.

minimum number of phases for any combination of link failures. Our algorithm only applies to the all-to-all traffic matrix.

We consider here a fat-tree where the root node (representing the set of spines in the generalized fat-tree topology) has M_1 children representing the leaf switches, and each leaf switch has M_0 children representing the servers. There are $P = M_0 M_1$ servers in total. Assuming the bandwidth between a server and its parent leaf switch is normalized to 1, the available bandwidth between a leaf switch and the root is M_0. Such a fat-tree is denoted $FT(2; M_0, M_1)$. There is a one to one mapping between the resources in this topology and the k-ary-L-tree deployed in practice. This new representation is used to simplify the upcoming notations.

Let f be the reduction of the bandwidth between each leaf switch and the root of the tree. In a generalized fat-tree, the root is replaced by M_0 spine switches, each connected to all the leaf switch links having equal bandwidth [34]. In this case, f is the maximum number of failed links on a leaf switch. The bandwidth reduction f is computed by taking the number of failures on all the leaf switches and finding the maximum of these numbers. It is known [18] that, if $M_0 \geq M_1$, then a schedule exists with a bandwidth usage between a leaf switch and the root of at most $M_0 - \lfloor \frac{M_0}{M_1} \rfloor$. So if $f \leq \lfloor \frac{M_0}{M_1} \rfloor$, the optimal schedule described in [18] works without being impacted by faults.

Now we present a schedule that performs an all-to-all (congestion-free) schedule for any $\lfloor \frac{M_0}{M_1} \rfloor < f < M_0$. The number of phases of our schedule is $P_f = \left\lceil \frac{M_0(P-M_0)}{M_0-f} \right\rceil > P$, which is optimal [65].

The index of a server $s \in [0, M_0 M_1]$ is uniquely decomposed as

$$s = \alpha_s + \gamma_s M_0$$

where $\gamma_s \in [0, M_1 - 1]$ is the index of the parent leaf switch and $\alpha_s \in [0, M_0 - 1]$ is the index of the server among the servers below the leaf switch.

We want each server to transmit $P - M_0$ times evenly during the P_f phases of an execution. These are the number of communications that need to leave the leaf switch, for each source server. To do so, we define for each server s the set of phases Θ_s when it transmits. The definition of Θ_s is based on a common infinite sequence $\tilde{\Theta}$ of evenly distributed integers with a density of $M_0/(M_0 - f)$:

$$\tilde{\Theta}(i) = \left\lceil \frac{iM_0}{M_0 - f} \right\rceil$$

where i is the index in the sequence.

We assign a shifted subset of the image of $\tilde{\Theta}$ of size $P - M_0$ to each server s as follows:

$$\Theta_s = \left\{ \tilde{\Theta}(i + \theta_s) - \alpha_s \text{ s.t. } i \in [0, P - M_0 - 1] \right\}$$

with $\theta_s = \left\lfloor \frac{(\alpha_s - 1)(M_0 - f)}{M_0} \right\rfloor + 1$. Θ_s contains $P - M_0$ integers from the sequence $\tilde{\Theta}$, shifted by α_s, and starting at index θ_s (in order ensure that the first phase is non-negative).

Let T be the *transmission function* defined on pairs (s, p) where $s \in [0, P - 1]$ is a server and $p \in [0, P_f - 1]$ a phase, such that $T(s, p)$ equals 1 if s transmits in phase p and 0 otherwise. In our schedule, we define T as follows:

$$T(s, p) = \begin{cases} 1 & \text{if } p \in \Theta_s \\ 0 & \text{otherwise} \end{cases}$$

We see that T is periodic with period M_0 and depends only on α_s and p. Figure 4 presents an example of a function T in a fat-tree $FT(2; 6, 6)$ with $f = 2$ failures.

Let $N_T(s, p)$ be the number of times s transmits before phase p (ie., $\sum_{q=0}^{p-1} T(s, q)$).

We can now define our schedule d. For a given server s in a phase p, if the server transmits at this phase, ie., if $T(s, p) = 1$, then the destination server is computed using the number $N_T(s, p) + \theta_s$. This number is used to compute the destination leaf D_{leaf} (among the $M_1 - 1$ possible destination leaves) and the index of the destination server below this leaf D_{server} (among the M_0 possible servers). The destination index $D_{leaf} M_0 + D_{server}$, which depends only on α_s, is then shifted to start from the index $(\gamma_s + 1)M_0$ of the first server below the next leaf. In addition, one could observe that the sequence of destinations is periodic with period the lowest common multiple between $M_1 - 1$ and M_0 (denoted $lcm(M_1 - 1, M_0)$, so if the $M_1 - 1$ and M_0 are not coprime, we must shift the sequence every $lcm(M_1 - 1, M_0)$ phase. Formally, we have:

$$\begin{cases} D_{server} = (N_T(s, p) + \theta_s) \,\%M_0 \\ D_{leaf} = \left(N_T(s, p) + \theta_s + \left\lfloor \frac{N_T(s,p)}{lcm(M_1 - 1, M_0)} \right\rfloor \right) \%(M_1 - 1) \\ d(s, p) = (D_{leaf} M_0 + D_{server} + (\gamma_s + 1)M_0) \,\%P \end{cases}$$

The scheduling algorithm is described in Algorithm 1. The scheduling solution can be found in polynomial time. The complexity of the scheduling algorithm depends on

the number of phases and the number of servers, and is given by $O(P_f * P)$, where $P_f = \left\lceil \frac{M_0(P-M_0)}{M_0-f} \right\rceil > P$ is the number of phases with f reduction of the bandwidth and $P = M_0 M_1$ the number of servers.

5 Fault-adaptive routing solution

In order to recover from failures, a better communication pattern is not enough, it is also necessary to ensure that the right paths are taken. Here we propose a congestion free routing solution to map the traffic resulting from the schedule. We first propose an algorithm that addresses link failures verifying a specific property. We then propose an ILP model to deal with all the failures not supported by Algorithm 2 in the next section.

5.1 Routing algorithm

The objective of our routing algorithm is to efficiently assign a unique spine to each source-destination pair during each phase. Indeed, selecting a spine dictates the whole path between a source and destination. For this purpose we go back to the original Generalized Fat-tree, more specifically the k-ary-L-tree [32]. To prevent multiple communication flows from utilizing the same link in the topology, the algorithm must guarantee that a spine is not used more than once for destinations connected to the same leaf switch as well as for sources below a leaf.

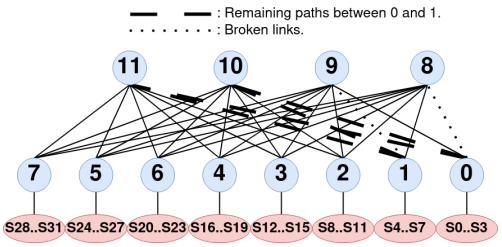

Fig. 5: Illustration of a failure scenario on a two-layer fat-tree topology involving an unequal number of paths between different leaf switches. The links 0-8 and 1-9 are broken, resulting in switch 0 having only 2 paths available to switch 1 and 3 paths available to the rest of the leaf switches.

We propose Algorithm 2 to compute a unique spine to each source-destination pair during each phase, ensuring that a spine is not used more than once for destinations or sources connected to the same leaf. The principle of Algorithm 2 is simply to exclude spines that contain at least one failure from being used as paths for source-destination pairs. This property makes Algorithm 2 only applicable when the number of spines impacted by failures m is inferior or equal to the bandwidth reduction f ($m \leq f$). For instance, in Figure 5 which represents a two-layer fat-tree, the bandwidth reduction f is 1, as the maximum number of link failures on leaf switches is 1. The number of

Algorithm 1 The scheduling algorithm computes the destination for each source at each phase. Here, M_1 is the number leaf switches, M_0 is the number of servers connected to a leaf switch and f is the bandwidth reduction. The output is *scheduling* which denotes the computed scheduling that gives the ordered list of source-destination pairs for each phase.

procedure *compute_scheduling*(M_0, M_1, f)
  $P \leftarrow M_0 * M_1$ ▷ Number of servers.
  $nbPhase \leftarrow \left\lceil \frac{(P-M_0)*M_0}{M_0-f} \right\rceil$ ▷ Number of phases.
  $\tilde{\Theta} \leftarrow []$ ▷ Computation of the sequence $\tilde{\Theta}$ of evenly distributed integers.
  for each $i \in [0..P]$ **do**
    $\tilde{\Theta}[i] \leftarrow \left\lceil \frac{iM_0}{M_0-f} \right\rceil$
  end for
  $\theta \leftarrow []$ ▷ Computation of the starting index from the sequence $\tilde{\Theta}$ for each server.
  for each $\alpha_s \in [0..M_0]$ **do**
    $\theta[\alpha_s] \leftarrow \left\lfloor \frac{(\alpha_s-1)(M_0-f)}{M_0} \right\rfloor + 1$
  end for
  $\Theta \leftarrow []$; ▷ Computation for each server of the set of phases Θ when it transmits.
  for each $\alpha_s \in [0..M_0]$ **do**
    $\Theta[\alpha_s] \leftarrow []$
    for each $i \in [0..P - M_0]$ **do**
      $\Theta[alpha_s][i] \leftarrow \tilde{\Theta}[i + \theta[\alpha_s]] - \alpha_s$
    end for
  end for
  $cycleLength \leftarrow lcm(M_1 - 1, M_0)$ ▷ Number of transmission before returning to the same destination. The lcm function returns the least common multiple between $M_1 - 1$ and M_0.
  $N_T \leftarrow [0] * P$ ▷ Number of times $N_T[s]$ a server s has transmitted data.
  $scheduling \leftarrow []$ ▷ Computation of the ouput *scheduling*.
  for each $p \in [0..nbPhase]$ **do**
    $scheduling[p] \leftarrow []$
    for each $sourceLeaf \in [0..M_1]$ **do**
      $sourceLeafOffset \leftarrow sourceLeaf * M_0$
      for each $sourceOffset \in [0..M_0]$ **do**
        $s \leftarrow sourceLeafOffset + sourceOffset$
        if $p \in \Theta[sourceOffset]$ **then**
          if $N_T[s] \geq P - M_0$ **then** ▷ The server has finished transmitting.
            continue
          end if
          $n \leftarrow N_T[s] + \theta[sourceOffset]$
          $cycleOffset \leftarrow \left\lfloor \frac{N_T(s,p)}{cycleLength} \right\rfloor$
          $groupDest \leftarrow n \mod (M_1 - 1)$
          $offsetDest \leftarrow (n + cycleOffset) \mod M_0$
          $shift \leftarrow groupDest * M_0 + offsetDest$
          $N_T[s] \leftarrow N_T[s] + 1$
          $d \leftarrow (sourceLeafOffset + M_0 + shift) \mod P$
          $scheduling[p].insert((s, d))$
        end if
      end for
    end for
  end for
end procedure

spines impacted by failures m is 2, as spines 8 and 9 each have one link failure with 0 and 1, respectively. As $m > f$, Algorithm 2 is not applicable in this scenario. However, if there is only one failure, involving switches 0 and 8 for example, the algorithm is applicable since the bandwidth reduction remains one and the number of spines impacted by failures is now one $m == f$ as well.

The spine to use on a path is chosen based on the index of the source-destination pair in the scheduling defined in Section 4. The selected spine is identified by the index i in Algorithm 2.

Consider a generalized fat-tree with $n = M_0$ spines and consider the array of n spines denoted $N = (N_0, \ldots, N_{n-1})$. Let $p \in P$ be a phase, P be the ordered list of phases and $D_l^p = (d_0^l, d_1^l, \ldots, d_{k-1}^l)$ be the ordered list of k servers (in increasing order) connected to leaf l that are destinations at phase p. The ordered list of $m \leq n$ spines not impacted by failures is denoted $S \in N$. A spine is impacted by failures if at least one of its connected links fails. The ordered list of source-destination pairs of servers that communicate at phase $p \in P$ is defined by $Schedule_p$ with p being the phase.

In the routing algorithm, each source-destination pair of servers denoted $(s, d) \in Schedule_p$ with $p \in P$, the spine used as a path to route the communication (s, d) is noted $S_{i \mod m} \in S$ with $i \mod m$ being the index of (s, d) in the ordered list $Schedule_p$. For example, $Schedule_p$ provides a set of source-destination pair $SD = ((s_0, d_0), \ldots, (s_m, d_m))$, m being the number of spines not impacted by failures and also the number of communications from and to a leaf switch at a phase $p \in P$. The spines $S_0, \ldots, S_m$ are respectively assigned as a path to the communications in SD, which means that S_0 is used as a path for the communication (s_0, d_0), S_1 is assigned to (s_1, d_1), etc.

Therefore, by definition, the source servers connected to a same leaf switch never use a same spine as the list is ordered (the source servers with the lowest ID are the first in the list) and the number of source server that communicates is equal to m. This ensures that there is no congestion between a source leaf and a spine. There is also no congestion between a spine and a destination leaf as the destinations in the scheduling algorithm defined in Section 4 are shifted.

The routing algorithm is described in Algorithm 2. The solution computed by the routing algorithm can be found in polynomial time. The complexity of the routing algorithm is the same as the complexity of the scheduling algorithm. The complexity is defined by $O(P_f * P)$, where $P_f = \left\lceil \frac{M_0(P-M_0)}{M_0-f} \right\rceil > P$ is the number of phases with f reduction of the bandwidth and $P = M_0 M_1$ is the number of servers.

However, in the case of other failure scenarios, in particular, if m spines are impacted by failures, with $m \leq f$, as illustrated in Figure 5, the routing problem becomes challenging. In order to be able to propose a general solution capable of addressing all failure scenarios without encountering congestion issues, our proposal is to use Integer Linear Programming (ILP).

5.2 Integer Linear programming model

In addressing specific failure scenarios, where the number of spines impacted by failures is strictly superior to the bandwidth reduction f, the formulation of a general

Algorithm 2 Routing algorithm to compute the spine for each source-destination pair (s, d) when m spines are impacted by failures and $m \leq f$, with f being the bandwidth reduction. Here, M_1 is the number leaf switches, M_0 is the number of servers connected to a leaf switch and S is the ordered list of spine switches that do not contain any failed links. *Schedule* is the scheduling computed by Algorithm 1 that provides the ordered list of source-destination pairs for each phase. The output is the dictionary *routingTable* which defines the computed spine for each source-destination pair (s, d).

procedure *compute_routing*$(M_0, M_1, f, S, Schedule)$
  $P \leftarrow M_0 * M_1$ ▷ Number of servers.
  $nbPhase \leftarrow \left\lceil \frac{(P-M_0)*M_0}{M_0-f} \right\rceil$ ▷ Number of phases.
  $routingTable \leftarrow \{\}$
  for each $p \in nbPhase$ **do**
    $i \leftarrow 0$
    for each $(s, d) \in Schedule_p$ **do**
      $sourceSwitch \leftarrow \lfloor \frac{s}{M_0} \rfloor$
      $destinationSwitch \leftarrow \lfloor \frac{d}{M_0} \rfloor$ ▷ If the communication (s, d) is not local to a leaf switch.
      if $sourceSwitch \neq destinationSwitch$ **then**
        $routingTable[(s, d)] \leftarrow S[i]$
        $i \leftarrow (i + 1) \mod |S|$
      end if
    end for
  end for
end procedure

routing algorithm appears, to our best understanding, unattainable. In response, our approach is to use Integer Linear Programming (ILP) to find a feasible routing solution, i.e. without conflicts on the links in the network. While using an ILP model to find optimal routing for a specific traffic matrix is not new and has been employed in the past, existing approaches [66, 67] solve scheduling and routing in a single model. This significantly increases the complexity of the problem and makes its applicability to large topologies challenging. In our approach, scheduling is solved by an algorithm in polynomial time, simplifying the routing problem. This significantly reduces the complexity of the ILP model for routing, making it applicable to large topologies.

We use Gurobi [68] to solve our Integer Linear Programming (ILP) model. Our choice is guided by the improved performance of Gurobi compared to other widely used solvers like IBM CPLEX and lpSolve [69]. An ILP model comprises variables, constraints, and an optimization function. However, in order to find a solution faster, and given that an optimization function is unnecessary as we seek only a feasible routing solution, we deliberately omitted such a function from our model and set the Gurobi parameters to stop after finding the first feasible solution. The notation to define the variables and constraints of the ILP model are described in Table A1.

Table 1: Notations in the ILP model.

Notation	Description
s	The source server.
d	The destination server.
S	The set of servers in the topology.
sp	The spine.
SP	The set of spines in the topology.
l	The leaf switch.
L	The set of leaf switches in the topology
S_l	The group of servers directly connected to the leaf switch l.

5.2.1 The variables

The variables determine which spine is used for each source-destination pair. They are defined as follow:

$v(s, d, sp)$: Binary variable where $s \in S$ is the source server, $d \in S$ is the destination server and $sp \in SP$ is the spine. There is a spine variable for each possible spine that can be used as a path for each source-destination pair. For example, if during a phase, the source server 0 uses the spine 8 to join the destination server 10, the variable $v(0, 10, 8)$ is equal to 1 in the ILP model; $\forall sp \neq 8, v(0, 10, sp) = 0$.

5.2.2 The constraints

We express the following constraints in the ILP model to ensure that the routing solution is conflict-free:

- Only one spine is chosen per communication:
  $\forall s, d \in S, \sum_{sp \in SP} v(s, d, sp) = 1$.
- At most one spine is used from each source leaf switch to avoid upstream congestion:
  $\forall l \in L, sp \in SP, \sum_{s \in S_l, d \in S} v(s, d, sp) \leq 1$.
- At most one spine is used to each destination leaf switch to avoid downstream congestion:
  $\forall l \in L, sp \in SP, \sum_{s \in S, d \in S_l} v(s, d, sp) \leq 1$.

One ILP model is run for each phase. The variables and constraints enable to find a feasible routing solution without conflicts on the network links for each phase independently.

6 Evaluation of performance

In this section, we evaluate the throughput achieved by FORS, our combined scheduling and routing solutions, on the use-case operational network with the settings described in Section 3.1. We compare FORS with state of the art solutions to prove that FORS degrades more gracefully in the event of bandwidth reduction than the other solutions. Furthermore, we illustrate the correlation between the increase in the number of phases and the achieved throughput, under various failure scenarios. Finally,

we present the computation time required to find a routing solution in complex failure scenarios, necessitating solving our ILP model.

6.1 Comparison of the achieved throughput

To evaluate our solution, we measure the throughput achieved by the servers under the various failure scenarios described in Section 3.1. The measurement methodology is exactly the same as described in Section 3.1. The InfiniBand controller assigns Local Identifiers (LIDs) to each server in the topology. In the switch routing tables, each LID is mapped to an output port, directing traffic accordingly. Our custom scheduling described in Section 4 is implemented in the MPI application deployed in the network. In instances of bandwidth reduction, FORS establishes a routing solution comprising multiple paths for each source-destination pair. The determination of the specific path for each pair depends on the phase in which the exchange occurs. Infiniband supports multipath [55], enabling the addition of multiple routes to the switch routing table for each destination server. Since the topology contains 20 spines, we need at least 20 routes for each LID in the routing tables of the switches. This means a switch needs both the LID of the server and the spine encoded in the LID to determine the appropriate output port. Thus, each switch has 20 LIDs to join each server, with every spine in the network having an assigned ID encoded within the LID. The parameter to set up multiple paths to join a destination server in InfiniBand is the LID Mask Control (LMC) [55]. The LMC value needed to implement the routing solution is 5, allowing at most $2^5 = 32$ routes for each destination server. We generate routing tables for each switch, where each LID is assigned to an output port. The InfiniBand controller then pushes the routing tables to the switches. Therefore, FORS scheduling, described in Section 4, assigns the LID to use for each communication at each phase, thus defining the complete route for sending traffic. This scheduling is implemented in the MPI application deployed in the network.

As shown in Figure 6, FORS outperforms all routing and scheduling solutions, significantly improving the throughput compared to these existing approach. Notably, FORS increases the throughput per server from 87-88 Gbps to 137 Gbps. With 326 servers in the DAQ network, this improvement results in a total network throughput increase from 28 Tbps to 44 Tbps.The throughput achieved by FORS decreases more gracefully than the other solutions when the bandwidth is reduced. Again, scenarios $1F\ SW_0$ and $1F\ SW_{0,5,11}$ exhibit the same throughput. This similarity is due to FORS being conflict-free and the exchanges being synchronized. Despite the additionnal failures in $1F\ SW_{0,5,11}$, $1F\ SW_0$ and $1F\ SW_{0,5,11}$ require the same number of phases for the all-to-all exchange. Consequently, the unique parameter that fluctuates the throughput is the number of phases, which increases upon failures to prevent conflicts. The higher number of phases increases the duration of the all-to-all completion time. This is true only for FORS. For the other solutions, the number of phases is unchanged. The throughput is determined by the number of conflicts that creates congestion which in turn fluctuates the throughput. The number of phases computed by FORS is influenced by the minimum bandwidth available between the leaf and spine switches. In both scenarios, the bandwidth is reduced by 1, resulting in an identical number of phases and similar results. The same applies to scenarios $2F\ SW_0$, $2F$

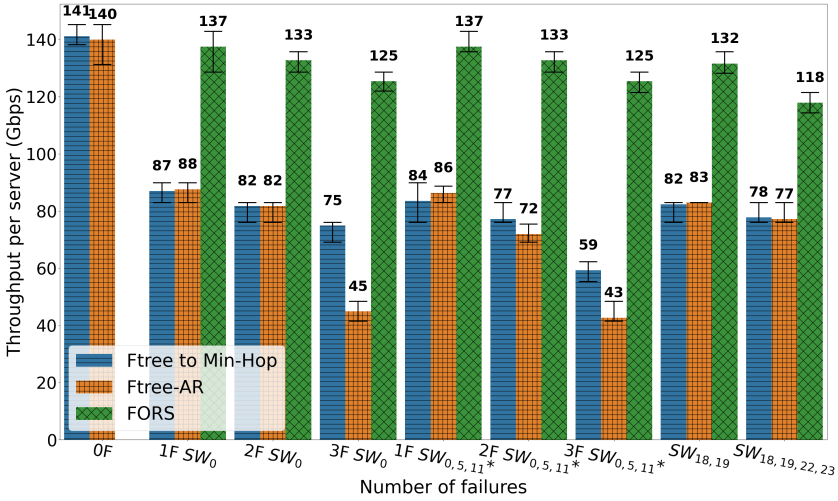

Fig. 6: Evaluation of FORS throughput compared to the default routing algorithms Ftree/Min-Hop and the adaptive routing version of Ftree. To obtain the total throughput achieved by the DAQ application, one can multiply the throughput per server by the number of servers in the DAQ application, which is 326. The error bars represent the minimum, mean and maximum values over one minute of measurement. The fat-tree topology of the studied data acquisition network is FT(2;20,18), illustrated in Figure 2. The failure scenarios noted with "*" are the ones where an ILP model was needed to compute the routing solution.

$SW_{0,5,11}$ and $SW_{18,19}$, where the bandwidth reduction is 2. This characteristic enables FORS to maintain the same throughput, in the case of two spine switches being lost (scenario $SW_{18,19}$) and the loss of two links on a single leaf switch (scenario $2F\ SW_0$).

6.2 Increase in the number of phases

Figure 7 illustrates the multiplicative factor I for the number of phases in the fat-tree topology, as a function of the bandwidth reduction. Both the fat-tree topology and the bandwidth reduction are specified. The vertical bar for each curve indicates a reduction of the bandwidth by half. The multiplicative factor I for the number of phases is the ratio of the number of phases with f bandwidth reduction over without bandwidth reduction: $I = \frac{P_f}{P_0}$. The bandwidth reduction corresponds to the maximum number of link failures on the leaf switches. This implies that the bandwidth reduction is the same for a single link failure and the loss of an entire spine, as the exchange is synchronized. Additionally, our approach being conflict-free, the only parameter affecting the throughput is the number of phases in the all-to-all scheduling which is optimal and dependent on the bandwidth reduction. As proved in Section 4, the optimal number of phases is computed by the formula: $P_f = \left\lceil \frac{M_0(P-M_0)}{M_0-f} \right\rceil$, where f is the bandwidth

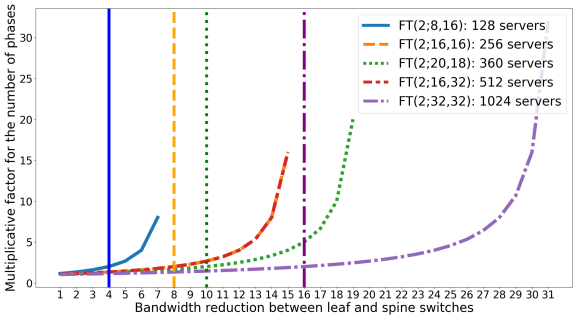

Fig. 7: Multiplicative factor for the number of phases as a function of the bandwidth reduction f between leaf and spine switches for several fat-tree topologies. FT(2;8,16) contains 128 servers, FT(2;16,16) contains 256 servers, FT(2;20,18), our use-case DAQ network, contains 360 servers, FT(2;16,32) contains 512 servers and FT(2;32,32) contains 1024 servers. The vertical lines correspond to the loss of half the bandwidth for each topology.

reduction between the leaf and spine switches, P is the number of servers and M_0 is the number of spine switches. Consequently, as the bandwidth reduction increases, the throughput decreases, but the decline is less pronounced compared to other solutions (as observed in Figure 6). This is because the increase in the number of phases remains optimal and does not significantly impact throughput when the bandwidth is reduced by less than half [65]. Furthermore, a scenario involving the loss of half the bandwidth or more in a network is unlikely.

We present the results for several fat-tree topologies: FT(2;8,16), FT(2;16,16), FT(2;20,18) (the topology of the use-case network), FT(2;16,32) and FT(2;32,32). For all topologies, the number of phases increases only slightly up to about half of bandwidth lost. For instance, this corresponds to $f = \frac{M_0}{2} = \frac{20}{2} = 10$ when half the bandwidth is lost for the topology FT(2;20,18). This increase in the number of phases proves sufficient to ensure the all-to-all exchange without conflicts even with half the bandwidth lost in the network. Consequently, FORS is able to compute a scheduling and routing solution with a number of phases only slightly higher than the number of phases without bandwidth reduction, resulting in a minimal drop in throughput.

6.3 Computation time

We measured the computation time of our solution under various randomly generated failure scenarios with the number of failures varying from 1 to the loss of half the links in the entire topology. We run Gurobi with a "11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz" processor, using 8 threads. The average computation time to find a routing solution for the topology of the studied DAQ network (FT(2;20,18)) is 29.72 seconds, with a minimum value of 27.399 seconds and a maximum value of 37.554 seconds. While the computation time needed to find a routing solution is not excessively long, it remains an inconvenience. However, these computation times do

not apply to all failure scenarios. In scenarios where the number of spines impacted by failures is strictly superior to the bandwidth reduction, a routing solution can be found in polynomial time, as they are computed by the routing algorithm described in 5.1. Furthermore, one potential approach is to pre-calculate a routing solution for all complex failure scenarios.

6.4 Operational constraints

To conduct our evaluation, we manually deactivated links and reproduced various failure scenarios in order to evaluate FORS and compare it with other possible approaches. This allowed us to demonstrate the potential of FORS. However, the implementation is currently limited by Infiniband's lack of flexibility in implementing new static routing tables in the event of failures. Currently, the routing tables generated by FORS are applied using the "file" routing engine, which loads the routing table from a file [55]. The file needs to be regenerated for each failure scenario. However, the challenge with this approach arises from the limitation that only reachable spines can be used as routes. If a failure occurs on a spine, FORS generates a new scheduling and associated routing tables, excluding the affected spine as a potential route. In the meantime, traffic will not be routed in the network, generating traffic loss. We hope this limitation to be lifted in future releases of Infiniband. Further, this limitation does not exist in Ethernet networks; therefore, FORS is readily deployable on these networks.

7 Related work

Optimization of the all-to-all collective exchange pattern has already been largely addressed and discussed. Numerous solutions have introduced scheduling algorithms that optimize bisection bandwidth [18, 20] or are applied to diverse network topologies [25] such as Butterfly [21, 25] and Dragonfly [19]. However, these approaches do not consider failure scenarios.

In Scheduling-Aware Routing solutions [70] [71] [72], the authors attempted to make the all-to-all communication schedule free of congestion by spreading the traffic out more evenly over time. They are able to limit the number of communications sharing a link in the all-to-all schedule to two. However, in this paper [65], the authors proved the optimal number of phases based on the remaining bandwidth in the network. Here, we are able to reach that number enabling us to achieve an optimal conflict-free all-to-all schedule.

Alternate routing strategies, such as adaptive routing [73] [74] [75], aim to balance the traffic dynamically to reduce congestion. Nevertheless, we measured the performance of such solutions in the studied data acquisition (DAQ) network and it revealed that adaptive routing is unsuitable for bursty traffic, offering minimal time for routing adjustments in the event of failures.

Another routing algorithm named wFatTree [76] is an adaptation of Ftree routing with advanced load-balancing. The authors showed that wFatTree distributes the congestion on the links more evenly than Ftree, but congestion is still present, while FORS is a congestion-free solution.

While optimal oblivious routing that often rely on Integer Linear Programming (ILP) techniques have already been proposed [77] [66], the solutions are hardly applicable for large-scale systems due to the computation time of the routing solution, and they do not handle the scheduling. Here we propose an algorithm with reduced complexity to solve the scheduling in all scenarios and the routing for basic failure scenarios. Furthermore, our proposed ILP model is able to solve the routing for specific failure scenarios in approximately 30 seconds.

A relevant approach to improve the bandwidth capacity in large-scale networks involves using demand-aware topologies [78] [79], which can be configured to match specific traffic patterns. However, in the context of the studied DAQ network, demand-aware topologies are not suitable, as the essential requirement of the DAQ application is a continuous utilization of all-to-all exchanges without involving any other traffic patterns. Consequently, in this paper, we focus on a static topology which is the fat-tree topology.

The fault-tolerance of the topology can further be improved by using the HyperX topology [80] [81] or F10 topology [82]. HyperX shows better throughput with the all-to-all exchange compared to the fat-tree topology [81]. F10 is a variation of a fat-tree that breaks its symmetry to provide more path diversity along with a routing algorithm. It remains to see whether a scalable scheduling algorithm can be proposed for such topologies.

8 Conclusion

In this paper, we address an observed gap that lies in the absence of scheduling and routing algorithms handling network failures for all-to-all traffic. This gap is important to address, as proved by our measurements, where utilizing a scheduling algorithm neglecting bandwidth reduction induces congestion in the studied data acquisition network. Consequently, the throughput of the DAQ network drops from 46 Tbps to 28 Tbps with a single failure.

In order to improve the performance in case of failures, we present FORS, a robust scheduling and routing solution designed to maintain high throughput in the event of failures on large-scale networks with a fat-tree topology and an all-to-all communication pattern. More specifically, we introduce a scheduling algorithm that produces congestion-free schedules for all-to-all exchanges and prove its optimality. Furthermore, we present a routing solution, composed of a simple algorithm and an Integer Linear Programming (ILP) model which, along with our scheduling approach, produces a congestion-free routing solution. The evaluation of our solution on a real large-scale Infiniband network demonstrated throughput improvement in the event of network failures compared to actual viable solutions in the studied network.

Notably, our solution achieves a total throughput of 44 Tbps in the presence of a single link failure, surpassing existing solutions that, at best, achieve 28 Tbps. Additionally, we demonstrate that FORS degrades more gracefully, experiencing only a minimal additional loss of a few Gbps as the bandwidth reduction increases. Finally, FORS demonstrates resilience by tolerating the failure of two switches, with throughput decreasing from 140 Gbps to 132 Gbps.

Appendix A Proof of the destination function in the scheduling algorithm

Table A1: Notations of the variables in the destination function.

Notation	Description
M_1	The number of leaf switches.
M_0	The number of servers.
$FT(2; M_0, M_1)$	A fat-tree topology with 2 layers.
f	The reduction of the bandwidth between each leaf switch and the root of the tree.
$P = M_0 M_1$	The number of servers and phases.
$P_f = \left\lfloor \frac{M_0(P-M_0)}{M_0-f} \right\rfloor > P$	The number of phases with bandwidth reduction.
$s \in [0, M_0 M_1] = \alpha_s + \gamma_s M_0$	The index of a server.
$\gamma_s \in [0, M_1 - 1]$	The index of the parent leaf switch.
$\alpha_s \in [0, M_0 - 1]$	The index of the server among the servers below the leaf switch.
$\tilde{\Theta}(s) = \left\lceil \frac{sM_0}{M_0-f} \right\rceil$	The set of phases when server s transmits.
$\Theta_s = \left\{ \tilde{\Theta}(i+\theta_s) - \alpha_s \text{ s.t. } i \in [0, P - M_0 - 1] \right\}$	Shifted subset of the image of $\tilde{\Theta}(s)$.
$\theta_s = \left\lfloor \frac{(\alpha_s - 1)(M_0 - f)}{M_0} \right\rfloor + 1$	Starting index from the sequence $\tilde{\Theta}_s$.
$T(s,p) = \begin{cases} 1 & \text{if } p \in \Theta_s \\ 0 & \text{otherwise} \end{cases}$	Transmission function defined on pairs (s,p) where $s \in [0, P-1]$ is a server and $p \in [0, P_f - 1]$ a phase.
$N_T(s,p)$	The number of times s transmits before phase p.
$D_{leaf} = \left(N_T(s,p) + \theta_s + \left\lfloor \frac{N_T(s,p)}{lcm(M_1-1, M_0)} \right\rfloor \right) \%(M_1 - 1)$	The destination leaf.
$D_{server} = (N_T(s,p) + \theta_s) \%M_0$	The index of the destination server below $leaf$.
$d(s,p) = \left(D_{leaf} M_0 + D_{server} + (\gamma_s + 1) M_0\right) \%P$	The schedule d for a given server s in a phase p, if the server transmits at this phase, ie., if $T(s,p) = 1$.

We prove that the destination function is well-defined and allows each server to transmit without congestion to all the other $P - M_0$ servers that are below different leaves. The notations are described in Table A1. We start by proving technical Lemmas about the sets Θ_s and the function N_T. Then, in Lemma 3 we prove that different servers have different destinations at each phase. In Lemma 4 we prove that a server never transmits two times to the same server. We show in Lemma 5 shows that our schedule creates no congestion and we conclude this section with our main Theorem 1.

Lemma 1. *$\forall s \in [0, P-1], \forall x \in \Theta_s$, then $x \in [0, P_f - 1]$.*

Proof. In this proof, we use the fact that for all $a \in \mathbb{Z}$, $a \leq \lceil a \rceil < a+1$ and $a-1 < \lfloor a \rfloor \leq a$. The smallest element in Θ_s is $x_{\min} = \tilde{\Theta}(\theta_s) - \alpha_s$ and we have

$$\begin{aligned} x_{\min} &= \left\lceil \left(\left\lfloor \frac{(\alpha_s - 1)(M_0 - f)}{M_0} \right\rfloor + 1 \right) \frac{M_0}{M_0 - f} \right\rceil - \alpha_s \\ &> \lceil \alpha_s - 1 \rceil - \alpha_s \Rightarrow x_{\min} \geq 0 \end{aligned}$$

The largest element is $x_{\max} = \tilde{\Theta}(P - M_0 - 1 + \theta_s) - \alpha_s$ and we have

$$\begin{aligned} x_{\max} &= \left\lceil \left(\left\lfloor \tfrac{(\alpha_s + P - M_0 - 2)(M_0 - f)}{M_0} \right\rfloor + 1 \right) \tfrac{M_0}{M_0 - f} \right\rceil - \alpha_s \\ &\leq \left\lceil P_f + (\alpha_s - 2) + \tfrac{M_0}{M_0 - f} \right\rceil - \alpha_s \\ &< P_f - 1 + \tfrac{M_0}{M_0 - f} \\ \Rightarrow x_{\max} &\leq P_f - 1 \end{aligned}$$

□

Lemma 2. *At a given phase, the difference between the number of transmissions for any two servers is at most one:*
$\forall s, s' \in [0, P-1]$, $\forall p \in [0, P_f - 1]$, $|N_T(s,p) - N_T(s',p)| \leq 1$.

Proof. First, we can observe that the number of phases between k consecutive transmissions of a given server is between $\lfloor (k-1)l \rfloor$ and $\lceil (k-1)l \rceil$, where $l = \frac{M_0}{M_0 - f}$. Indeed, for all $i \in \mathbb{N}$, we have

$$\Theta(i + k - 1) - \Theta(i) = \lceil (i + k - 1)l \rceil - \lceil il \rceil$$

It follows that

$$\begin{aligned} (i+k-1)l - 1 - il < \Theta(i+1) - \Theta(i) &< (i + k - 1)l - il + 1 \\ \lfloor (k-1)l \rfloor \leq (\Theta(i+1) - \Theta(i)) &\leq \lceil (k-1)l \rceil \end{aligned}$$

Now consider for the sake of contradiction that the difference in the number of transmitted messages is greater than one for two servers s and s', and let p'_2 be the first phase when this occurs, ie., s' transmits and $N_T(s', p'_2) = N_T(s, p'_2) + 2$.

From the previous observation, s' performs $k = N_T(s', p'_2)$ transmissions between phase 0 and p'_2 and $p'_2 \geq \lfloor kl \rfloor$

Let $p_2 \geq p'_2 + 2$ be the first phase after p'_2 such that $N_T(s, p_2) = N_T(s', p'_2) = k$. Again using the previous observation we obtain the following contradiction

$$\lceil kl \rceil \geq p_2 \geq p'_2 + 2 \geq \lfloor kl \rfloor + 2 \geq \lceil kl \rceil + 1$$

□

Remark 1. *In the remaining proofs, we use the following fact: for any integers a, b, c, if $a\%c = b\%c$ and $|a - b| < c$, then $a = b$.*

Lemma 3. *If $T(s,p) = T(s',p) = 1$ and $d(s,p) = d(s',p)$ for some $s, s' \in [0, P-1]$, then $s = s'$.*

Proof. Assume that $d(s,p) = d(s',p)$ for some $s, s' \in [0, P-1]$. Observe that M_0 divides P and we have $d(s,p) \equiv D_{server} \pmod{M_0}$[1]. So, $d(s,p) = d(s',p)$ implies that

$$N_T(s,p) + \theta_s \equiv N_T(s',p) + \theta_{s'} \pmod{M_0}. \tag{A1}$$

The following Claims help us prove that $\alpha_s = \alpha_{s'}$.

For simplicity, we define, for $p \geq 0$, $\mathcal{T}(p)$, resp. $\mathcal{N}_T(p)$, the transmission function of server 0, resp. the number of transmissions of server 0 before phase p, if server 0 continues to transmit using the same pattern after phase P_f. More formally, $\mathcal{T}(p) = 1$ if and only if $\exists i \geq 0, \tilde{\Theta}(i) = p$, and $\mathcal{N}_T(p) = \sum_{i=0}^{p-1} \mathcal{T}(p)$

Claim 1 $T(s,p) = \mathcal{T}(p + \alpha_s)$.

Proof: We have the following equivalences:

$$\begin{aligned} T(s,p) = 1 &\Leftrightarrow \exists i \in [0, P - M_0 - 1], p = \tilde{\Theta}(i + \theta_s) - \alpha_s \\ &\Leftrightarrow \exists i \in [0, P - M_0 - 1], p + \alpha_s = \tilde{\Theta}(i + \theta_s) \\ &\Leftrightarrow \mathcal{T}(p + \alpha_s) = 1 \end{aligned}$$

Claim 2 $N_T(s,p) = \mathcal{N}_T(p + \alpha_s) - \mathcal{N}_T(\alpha_s)$

Proof:

$$\begin{aligned} N_T(s,p) &= \sum_{i=0}^{p-1} T(s,i) = \sum_{i=0}^{p-1} \mathcal{T}(i + \alpha_s) \\ &= \sum_{i=0}^{p+\alpha_s-1} \mathcal{T}(i) - \sum_{i=0}^{\alpha_s-1} \mathcal{T}(i) \\ &= \mathcal{N}_T(p + \alpha_s) - \mathcal{N}_T(\alpha_s) \end{aligned}$$

Claim 3 $\theta_s = \mathcal{N}_T(\alpha_s)$.

Proof: Observe that $\mathcal{N}_T(\alpha_s)$ is the number of transmissions performed by server 0 before phase α_s, so it is the cardinal of the set $\Theta_0 \cap [0, \alpha_s - 1]$, so it is the number of integers i such that $\tilde{\Theta}(i) \leq \alpha_s - 1$. Since $\tilde{\Theta}$ is a strictly increasing function, this is equivalent to finding the first integer i such that $\tilde{\Theta}(i) \geq \theta_s$.

We already saw in the proof of Lemma 1 that $\tilde{\Theta}(\theta_s) \geq \alpha_s$, so it remains to show that $\tilde{\Theta}(\theta_s - 1) < \alpha_s$. We have

$$\tilde{\Theta}(\theta_s - 1) = \left\lceil \left\lfloor \frac{(\alpha_s - 1)(M_0 - f)}{M_0} \right\rfloor \frac{M_0}{M_0 - f} \right\rceil$$

[1] Recall that $a \equiv b \pmod{c}$ is equivalent to $a - b$ is divisible by c

$$\leq \lceil \alpha_s - 1 \rceil < \alpha_s$$

Claim 4 $\mathcal{N}_T(p + \alpha_s) = \mathcal{N}_T(p + \alpha_{s'})$.
Proof: From the two previous claims we have

$$N_T(s, p) + \theta_s = \mathcal{N}_T(p + \alpha_s)$$

From Equation A1 and the fact that $|\mathcal{N}_T(p + \alpha_s) - \mathcal{N}_T(p + \alpha_{s'})| \leq M_0 - f < M_0$ (because $|\alpha_s - \alpha_{s'}| < M_0$), then, using Remark 1, the claim is proved.

Claim 5 $\alpha_s = \alpha_{s'}$.
Proof: From the previous claim we know $\mathcal{N}_T(p + \alpha_s) = \mathcal{N}_T(p + \alpha_{s'})$. Since $T(s, p) = T(s', p) = 1$ we have $\mathcal{T}(p + \alpha_s) = \mathcal{T}(p + \alpha_s) = 1$ (Claim 1), hence

$$\mathcal{N}_T(p + \alpha_s + 1) = \mathcal{N}_T(p + \alpha_{s'} + 1) = \mathcal{N}_T(p + \alpha_s) + 1.$$

Clearly $\mathcal{N}_T$ is a non-decreasing function so $p + \alpha_s = p + \alpha_{s'}$ which implies $\alpha_s = \alpha_{s'}$.
Remaining Proof of the Lemma: Since D_{leaf} and D_{server} depend only on α_s (and not on γ_s) and $\alpha_s = \alpha_{s'}$, then

$$((\gamma_s + 1)M_0)\%P = ((\gamma_{s'} + 1)M_0)\%P$$

Also, $\gamma_s \in [0, M_1 - 1]$ implies that $(\gamma_s + 1)M_0 \in [M_0, P]$ so that

$$|(\gamma_s + 1)M_0 - (\gamma_{s'} + 1)M_0| < P$$

so, from Remark 1, we obtain

$$A + (\gamma_s + 1)M_0 = A + (\gamma_{s'} + 1)M_0 \Rightarrow \gamma_s = \gamma_{s'}$$

so that $s = s'$. □

Lemma 4. *If $T(s, p) = T(s, p') = 1$ and $d(s, p) = d(s, p')$ for some $p, p' \in [0, P_f - 1]$, then $p = p'$.*

Proof. As in the previous Lemma, assuming $d(s, p) = d(s, p')$ implies

$$\begin{aligned} &N_T(s, p) \equiv N_T(s, p') \pmod{M_0} \\ \Rightarrow &\exists k_0,\ N_T(s, p) - N_T(s, p') = k_0 M_0 \end{aligned}$$

Also, by dividing by M_0 both sides of the equality $d(s, p) = d(s, p')$, we obtain (with $L = \text{lcm}(M_1 - 1, M_0)$)

$$\begin{aligned} &N_T(s,p) + \left\lfloor \tfrac{N_T(s,p)}{L} \right\rfloor \equiv N_T(s,p') + \left\lfloor \tfrac{N_T(s,p')}{L} \right\rfloor \pmod{M_1 - 1} \\ \Rightarrow &\exists k_1,\ N_T(s,p) - N_T(s,p') = \left\lfloor \tfrac{N_T(s,p')}{L} \right\rfloor - \left\lfloor \tfrac{N_T(s,p)}{L} \right\rfloor + k_1(M_1 - 1) \end{aligned}$$

This gives the following Bézout's identity:

$$k_0 M_0 + k_1(M_1 - 1) = \left\lfloor \frac{N_T(s,p')}{L} \right\rfloor - \left\lfloor \frac{N_T(s,p)}{L} \right\rfloor$$

that have solutions only if the right side is a multiple of $\gcd(M_0, M_1 - 1)$. But one can see that, since $N_T(s,p') < M_0(M_1 - 1) = L \times \gcd(M_0, M_1 - 1)$, then the right side of the equation is smaller than $\gcd(M_0, M_1 - 1)$, so it is null, and $k_0 = k_1 = 0$. So $N_T(s,p) = N_T(s,p')$, which in turns gives $p = p'$. □

In the remaining, we call a *group* of servers, the set of servers below a given leaf switch. More precisely, a server s is in group g if $\gamma_s = g$.

Lemma 5. *At most $M_0 - f$ servers in the same group transmit at the same time. At most $M_0 - f$ servers in the same group are destinations at the same time.*

Proof. The servers in a group g are exactly the servers s with id $\alpha_s + gM_0$. By definition of T, the number of such servers that transmit at a given phase p is equal to the number of times the server with id $\gamma_s M_0$ (*i.e.*, the first server in the group) transmits in the next M_0 phases. By construction, this number is $M_0 - f$. Since this is true for each leaf, in total, there is at most $M_1(M_0 - f)$ senders at each phase.

About the destinations, assume for the sake of contradiction that $k > M_0 - f$ servers below a given leaf, with id of the form $i + gM_0$, are destinations at a given phase p. let r be such a destination server and let s be the sender of the transmission received by r. It is clear that the server $s' = \alpha_s + (\gamma_s + 1)M_0 \mod P$ have destination $r' = \alpha_r + (\gamma_r + 1)M_0 \mod P$. This implies that there are k servers with id of the form $i + (g+1)M_0$ that are destinations at phase p. Hence below every leaf, more than $M_0 - f$ servers are destinations. This is a contradiction as it implies that there are more than $M_1(M_0 - f)$ receivers, which is more than the number of senders. □

We are now ready to present our all-to-all schedule. The schedule d is well-defined for all the communications between servers of different groups. The communications between servers of the same group can be scheduled arbitrarily at each phase, between the servers that do not transmit using d. Let g be any schedule such that, at each phase p, at least f servers in each group that do not transmit according to d, transmit to other servers in the same group that are not receivers, until the servers in the same groups have exchanged all their messages.

Theorem 1. *The schedule defined by d for the communication between each server and the other servers outside its group, and by g for the communications between servers in the same group, is a congestion-free all-to-all schedule.*

Proof. From Lemma 3 and Lemma 4, we know that no collision occurs, *i.e.*, no two different servers transmit at the same time to the same server, and that each server never transmits two times to the same server. Lemma 5 proves that our schedule creates no congestion.

Since, using d, a server transmits $P - M_0$ times during the P_f phases of the execution (and each destination is different), then all the servers are reached, except the ones that are in the same group, which are reached using schedule g as explained below.

As we discussed at the beginning of the section, we consider $f > \left\lfloor \frac{M_0}{M_1} \right\rfloor$. Since f servers in a group can communicate at each phase and there are P_f phases, at least fP_f communications can occur concurrently with the schedule d. In a group, M_0^2 communication must be performed in total, so we have to verify that $fM_f \geq M_0^2$ which is equivalent to

$$f\left\lceil \frac{M_0(P-M_0)}{M_0-f} \right\rceil \frac{1}{M_0^2} \geq 1 \quad \Leftarrow \quad f\frac{M_1-1}{M_0-f} \geq 1$$

which follows from our initial assumption $f > \left\lfloor \frac{M_0}{M_1} \right\rfloor$.

□

Appendix B The spine pinning problem

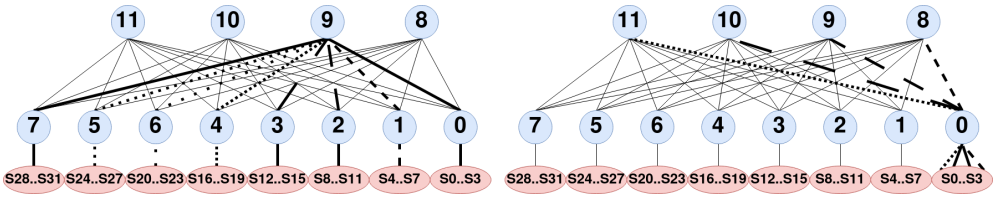

Fig. B1: Illustration of the deterministic nature of the paths in a two-layer fat-tree topology. On the left, all servers have one unique path to join spine 9 represented by different dashed lines. On the right, all spine switches have one unique path to join servers 0..3.

In a fat-tree topology, routing can be simplified by selecting a spine for each source-destination pair, giving us the complete path for routing the communication flow. In a two-layer fat-tree topology, as in the studied DAQ network, the leaf switches are directly connected to the spines, making this statement evident. However, this properly extends to fat-tree topologies with more than two layers as well. In a k-ary-L-tree [32] fat-tree topology, the shortest path to go from a spine to a destination server is always unique. Figure B1 represents a two-layer fat-tree topology, as illustrated, all servers have one unique path to join spine 9 and all spine switches have one unique path to join servers 0..3. This property is valid on any type of k-ary-L-tree fat-tree, even in topologies with more than 2 layers. The path from the spines to a server remains deterministic, and vice versa, regardless of the scale of the network.

This property makes the routing problem easy to solve with a straightforward algorithm explained in Section 5.1, when the bandwidth reduction is uniform across all leaf switches or when there is no reduction in bandwidth. However, the routing problem becomes more challenging to solve in specific failure scenarios that involve unequal bandwidth reduction between leaf switches, as introduced in Section 3.2.

Figure 5 illustrates a two-layer fat-tree topology with a failure scenario involving an unequal number of paths between different leaf switches. In the event of two link

failures in this topology, specifically, one between switches 0 and 8 and another between switches 1 and 9, the number of paths available between switches 0 and 1 is reduced to 2, see long dashes in Figure 5. Whereas all other leaf switches maintain 3 paths available to communicate with switches 0 and 1. This limitation implies that if, during a single phase, there are more than two communications between switches 0 and 1, congestion is unavoidable. To address such failure scenarios, our proposed scheduling algorithm, described in Section 4, always produces a schedule where the number of communications between two leaf switches cannot be greater than the number of spines. Consequently, as long as there is at least one path between two leaf-switches, the scheduling produced by our algorithm ensures that a routing solution is possible. With all these considerations, the complexity of the routing problem is significantly reduced to choosing a single spine for each pair of communications at each phase. The scheduling algorithm we have implemented introduces a key property, enabling us to resolve all failure scenarios, as long as there is a path available between two servers.

References

[1] Borrill, J., Keskitalo, R., Kisner, T.: Big Bang, Big Data, Big Iron: Fifteen Years of Cosmic Microwave Background Data Analysis at NERSC. Computing in Science & Engineering **17**(3), 22–29 (2015) https://doi.org/10.1109/MCSE.2015.1

[2] Dorelli, J., Bard, C., Chen, T., Silva, D., Santos, L.F., Ireland, J., Kirk, M., McGranaghan, R., Narock, A., Nieves-Chinchilla, T., Samara, M., Sarantos, M., Schuck, P., Thompson, B.: Deep Learning for Space Weather Prediction: Bridging the Gap between Heliophysics Data and Theory (2022)

[3] Update, P.: SIMULIA. [Online]. Available: https://www.fsb.unizg.hr/atlantis/upload/newsboard/22_07_2011__15351_SIMULIA_RSN-May2011.pdf. Accessed on: June 24, 2024. (2011)

[4] Liu, X., Xu, Z., Liu, G., Liu, L.: Intelligent Compound Selection of Anti-cancer Drugs Based on Multi-Objective Optimization. In: 2023 International Conference on Intelligent Supercomputing and BioPharma (ISBP), pp. 48–53. IEEE, Zhuhai, China (2023). https://doi.org/10.1109/ISBP57705.2023.10061321

[5] Leung, C.K., Sarumi, O.A., Zhang, C.Y.: Predictive Analytics on Genomic Data with High-Performance Computing. In: 2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 2187–2194. IEEE, Seoul, Korea (South) (2020). https://doi.org/10.1109/BIBM49941.2020.9312982

[6] Belyaev, N., Krasnopevtsev, D., Konoplich, R., Velikhov, V., Klimentov, A.: High performance computing system in the framework of the Higgs boson studies. Technical report, ATL-COM-SOFT-2017-089 (2017)

[7] Jereczek, G., Lehmann Miotto, G., Malone, D.: Analogues between tuning TCP

for Data Acquisition and datacenter networks. In: 2015 IEEE International Conference on Communications (ICC), pp. 6062–6067. IEEE, London, UK (2015). https://doi.org/10.1109/ICC.2015.7249288

[8] Bawej, T., Behrens, U., Branson, J., Chaze, O., Cittolin, S., Darlea, G.-L., Deldicque, C., Dobson, M., Dupont, A., Erhan, S., Forrest, A., Gigi, D., Glege, F., Gomez-Ceballos, G., Gomez-Reino, R., Hegeman, J., Holzner, A., Masetti, L., Meijers, F., Meschi, E., Mommsen, R.K., Morovic, S., Nunez-Barranco-Fernandez, C., O'Dell, V., Orsini, L., Paus, C., Petrucci, A., Pieri, M., Racz, A., Sakulin, H., Schwick, C., Stieger, B., Sumorok, K., Veverka, J., Zejdl, P.: The New CMS DAQ System for Run-2 of the LHC. IEEE Transactions on Nuclear Science **62**(3), 1099–1103 (2015) https://doi.org/10.1109/TNS.2015.2426216

[9] LHCb Collaboration: LHCb Trigger and Online Upgrade Technical Design Report. Technical report, CERN, Geneva (2014)

[10] LHCb Collaboration: LHCb Upgrade GPU High Level Trigger Technical Design Report. Technical report, CERN, Geneva (2020). https://doi.org/10.17181/CERN.QDVA.5PIR

[11] CERN: Key Facts and Figures – CERN Data Centre. [Online]. Available: https://information-technology.web.cern.ch/sites/default/files/CERNDataCentre_KeyInformation_July2019V1.pdf. Accessed on: June 24, 2024. (2019)

[12] Wu, S., Zhai, Y., Liu, J., Huang, J., Jian, Z., Wong, B., Chen, Z.: Anatomy of High-Performance GEMM with Online Fault Tolerance on GPUs. In: Proceedings of the 37th International Conference on Supercomputing. ICS '23, pp. 360–372. Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3577193.3593715 . https://doi.org/10.1145/3577193.3593715

[13] Wang, R., Dong, D., Lei, F., Ma, J., Wu, K., Lu, K.: Roar: A Router Microarchitecture for In-network Allreduce. In: Proceedings of the 37th International Conference on Supercomputing. ICS '23, pp. 423–436. Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3577193.3593711 . https://doi.org/10.1145/3577193.3593711

[14] Chirkov, G., Wentzlaff, D.: Seizing the Bandwidth Scaling of On-Package Interconnect in a Post-Moore's Law World. In: Proceedings of the 37th International Conference on Supercomputing. ICS '23, pp. 410–422. Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3577193.3593702 . https://doi.org/10.1145/3577193.3593702

[15] Huang, J., Di, S., Yu, X., Zhai, Y., Liu, J., Huang, Y., Raffenetti, K., Zhou, H., Zhao, K., Chen, Z., et al.: gzccl: Compression-accelerated collective communication framework for gpu clusters. arXiv preprint arXiv:2308.05199 (2023)

[16] Contini, N., Ramesh, B., Kandadi Suresh, K., Tran, T., Michalowicz, B., Abduljabbar, M., Subramoni, H., Panda, D.: Enabling Reconfigurable HPC through MPI-based Inter-FPGA Communication. In: Proceedings of the 37th International Conference on Supercomputing. ICS '23, pp. 477–487. Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3577193.3593720 . https://doi.org/10.1145/3577193.3593720

[17] Feng, G., Dong, D., Zhao, S., Lu, Y.: GRAP: Group-Level Resource Allocation Policy for Reconfigurable Dragonfly Network in HPC. In: Proceedings of the 37th International Conference on Supercomputing. ICS '23, pp. 437–449. Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3577193.3593732

[18] Prisacari, B., Rodriguez, G., Minkenberg, C., Hoefler, T.: Bandwidth-Optimal All-to-All Exchanges in Fat Tree Networks. In: Proceedings of the 27th International ACM Conference on International Conference on Supercomputing. ICS '13, pp. 139–148. Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2464996.2465434

[19] Prisacari, B., Rodriguez, G., Minkenberg, C.: Generalized Hierarchical All-to-All Exchange Patterns. In: 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, pp. 537–547. IEEE, Cambridge, MA, USA (2013). https://doi.org/10.1109/IPDPS.2013.87

[20] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., Vahdat, A.: Hedera: Dynamic Flow Scheduling for Data Center Networks. NSDI'10, p. 19. USENIX Association, USA (2010)

[21] Izzi, D., Massini, A.: Optimal all-to-all personalized communication on Butterfly networks through a reduced Latin square. In: 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 1065–1072. IEEE, Yanuca Island, Cuvu, Fiji (2020). https://doi.org/10.1109/HPCC-SmartCity-DSS50907.2020.00195

[22] Zahavi, E., Johnson, G., Kerbyson, D., Lang, M.: Optimized InfiniBandTM fat-tree routing for shift all-to-all communication patterns. Concurrency and Computation: Practice and Experience **22**, 217–231 (2009) https://doi.org/10.1002/cpe.1527

[23] Peng, J., Liu, J., Dai, Y., Xie, M., Gong, C.: Optimizing All-to-All Collective Communication on Tianhe Supercomputer. In: 2022 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), pp. 402–409. IEEE, Melbourne, Australia (2022). https://doi.org/10.1109/

ISPA-BDCloud-SocialCom-SustainCom57177.2022.00058

[24] Izzi, D., Massini, A.: All-to-All Personalized Communication on Fat-Trees Using Latin Squares. In: 2022 International Conference on Software, Telecommunications and Computer Networks (SoftCOM), pp. 1–6. IEEE, Split, Croatia (2022). https://doi.org/10.23919/SoftCOM55329.2022.9911285

[25] Izzi, D., Massini, A.: Realizing Optimal All-to-All Personalized Communication Using Butterfly-Based Networks. IEEE Access **11**, 51064–51083 (2023) https://doi.org/10.1109/ACCESS.2023.3279494

[26] Singh, R., Mukhtar, M., Krishna, A., Parkhi, A., Padhye, J., Maltz, D.: Surviving switch failures in cloud datacenters. SIGCOMM Comput. Commun. Rev. **51**(2), 2–9 (2021) https://doi.org/10.1145/3464994.3464996

[27] Sergeev, A., Del Balso, M.: Horovod: fast and easy distributed deep learning in TensorFlow. arXiv preprint arXiv:1802.05799 (2018)

[28] Zhao, L., Maleki, S., Yang, Z., Pourreza, H., Shah, A., Hwang, C., Krishnamurthy, A.: ForestColl: Efficient Collective Communications on Heterogeneous Network Fabrics. arXiv preprint arXiv:2402.06787 (2024)

[29] Zhou, Q., Anthony, Q., Xu, L., Shafi, A., Abduljabbar, M., Subramoni, H., Panda, D.K.D.: Accelerating Distributed Deep Learning Training with Compression Assisted Allgather and Reduce-Scatter Communication. In: 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 134–144 (2023). https://doi.org/10.1109/IPDPS54959.2023.00023

[30] Wang, W., Ghobadi, M., Shakeri, K., Zhang, Y., Hasani, N.: Optimized Network Architectures for Large Language Model Training with Billions of Parameters. arXiv preprint arXiv:2307.12169 (2023)

[31] Nvidia: Collective Operations. [Online]. Available: https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/collectives.html. Accessed on: June 24, 2024. (2020)

[32] Petrini, F., Vanneschi, M.: k-ary n-trees: high performance networks for massively parallel architectures. In: Proceedings 11th International Parallel Processing Symposium, pp. 87–93. IEEE, Geneva, Switzerland (1997). https://doi.org/10.1109/IPPS.1997.580853

[33] List, T.: TOP500 List. [Online]. Available: https://www.top500.org/lists/top500/2024/06/. Accessed on: June 24, 2024. (2024)

[34] Pippenger, N.: On rearrangeable and non-blocking switching networks. Journal of Computer and System Sciences **17**(2), 145–162 (1978) https://doi.org/10.1016/0022-0000(78)90001-6

[35] Yao, F., Wu, J., Venkataramani, G., Subramaniam, S.: A comparative analysis of data center network architectures. In: 2014 IEEE International Conference on Communications (ICC), pp. 3106–3111. IEEE, Sydney, NSW, Australia (2014). https://doi.org/10.1109/ICC.2014.6883798

[36] Sen, A., Datta, A., De, M.: Fault Tolerant Wormhole Routing for Complete Exchange in Multi-Mesh. In: 2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS), pp. 415–420. IEEE, Belgaum, India (2018). https://doi.org/10.1109/CTEMS.2018.8769323

[37] Yazaki, S., Takaue, H., Ajima, Y., Shimizu, T., Ishihata, H.: An Efficient All-to-all Communication Algorithm for Mesh/Torus Networks. In: 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications, pp. 277–284. IEEE, Leganes, Spain (2012). https://doi.org/10.1109/ISPA.2012.44

[38] Doi, J., Negishi, Y.: Overlapping Methods of All-to-All Communication and FFT Algorithms for Torus-Connected Massively Parallel Supercomputers. In: SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–9. IEEE, New Orleans, LA, USA (2010). https://doi.org/10.1109/SC.2010.38

[39] Kim, J., Dally, W.J., Scott, S., Abts, D.: Technology-Driven, Highly-Scalable Dragonfly Topology. In: 2008 International Symposium on Computer Architecture, pp. 77–88 (2008). https://doi.org/10.1109/ISCA.2008.19

[40] Shpiner, A., Haramaty, Z., Eliad, S., Zdornov, V., Gafni, B., Zahavi, E.: 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), pp. 1–8. IEEE, Austin, TX, USA (2017). https://doi.org/10.1109/HiPINEB.2017.11

[41] Beni, M.S., Cosenza, B.: An Analysis of Performance Variability on Dragonfly+topology. In: 2022 IEEE International Conference on Cluster Computing (CLUSTER), pp. 500–501. IEEE, Heidelberg, Germany (2022). https://doi.org/10.1109/CLUSTER51413.2022.00061

[42] MPI: MPI(3) man page. [Online]. Available: https://www.open-mpi.org/doc/v4.0/man3. Accessed on: June 24, 2024. (2021)

[43] Dalcin, L., Mortensen, M., Keyes, D.E.: Fast parallel multidimensional FFT using advanced MPI. Journal of Parallel and Distributed Computing **128**, 137–150 (2019) https://doi.org/10.1016/j.jpdc.2019.02.006

[44] Czechowski, K., Battaglino, C., McClanahan, C., Iyer, K., Yeung, P.-K., Vuduc, R.: On the Communication Complexity of 3D FFTs and Its Implications for Exascale. In: Proceedings of the 26th ACM International Conference on

Supercomputing. ICS '12, pp. 205–214. Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2304576.2304604 . https://doi-org.ezproxy.cern.ch/10.1145/2304576.2304604

[45] Sumanaweera, T., Liu, D.: Medical image reconstruction with the FFT. GPU gems **2**, 765–784 (2005)

[46] Müller, A., Deconinck, W., Kühnlein, C., Mengaldo, G., Lange, M., Wedi, N., Bauer, P., Smolarkiewicz, P., Diamantakis, M., Lock, S.-J., Saarinen, S., Mozdzynski, G., Thiemert, D., Glinton, M., Bénard, P., Voitus, F., Colavolpe, C., Marguinaud, P., New, N.: The ESCAPE project: Energy-efficient Scalable Algorithms for Weather Prediction at Exascale. Geoscientific Model Development **12**, 4425–4441 (2019) https://doi.org/10.5194/gmd-12-4425-2019

[47] Roy, A., Zeng, H., Bagga, J., Porter, G., Snoeren, A.C.: Inside the Social Network's (Datacenter) Network. In: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. SIGCOMM '15, pp. 123–137. Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2785956.2787472 . https://doi.org/10.1145/2785956.2787472

[48] Jereczek, G.: Software switching for high throughput data acquisition networks. PhD thesis, National University of Ireland, Maynooth (Ireland) (2017)

[49] Amoiridis, V., James, T.O., Rabady, D.S., Zogatova, D., Gigi, D., Racz, A., Deldicque, C., Cano, E., Meschi, E., Brummer, P.M., et al.: The cms orbit builder for the hl-lhc at cern. Technical report, CERN (2023)

[50] Pisani, F., Colombo, T., Durante, P., Frank, M., Gaspar, C., Cardoso, L.G., Neufeld, N., Perro, A.: Design and Commissioning of the First 32-Tbit/s EventBuilder. IEEE Transactions on Nuclear Science **70**(6), 906–913 (2023) https://doi.org/10.1109/TNS.2023.3240514

[51] CERN: The Large Hadron Collider. [Online]. Available: https://home.cern/science/accelerators/large-hadron-collider. Accessed on: June 24, 2024. (2024)

[52] Pisani, F., Colombo, T., Durante, P., Frank, M., Gaspar, C., Cardoso, L.G., Neufeld, N., Perro, A.: Design and Commissioning of the First 32-Tbit/s EventBuilder. IEEE Transactions on Nuclear Science **70**(6), 906–913 (2023) https://doi.org/10.1109/TNS.2023.3240514

[53] Amoiridis, Vassileios, Behrens, Ulf, Bocci, Andrea, Branson, James, Brummer, Philipp, Cano, Eric, Cittolin, Sergio, Da Silva Almeida Da Quintanilha, Joao, Darlea, Georgiana-Lavinia, Deldicque, Christian, Dobson, Marc, Dvorak, Antonin, Gigi, Dominique, Glege, Frank, Gomez-Ceballos, Guillelmo, Gorniak, Patrycja, Gutić, Neven, Hegeman, Jeroen, Izquierdo Moreno, Guillermo, James, Thomas Owen, Karimeh, Wassef, Kartalas, Miltiadis, Krawczyk, Rafał Dominik, Li, Wei, Long, Kenneth, Meijers, Frans, Meschi, Emilio, Morović, Srećko, Orsini,

Luciano, Paus, Christoph, Petrucci, Andrea, Pieri, Marco, Rabady, Dinyar Sebastian, Racz, Attila, Rizopoulos, Theodoros, Sakulin, Hannes, Schwick, Christoph, Šimelevičius, Dainius, Tzanis, Polyneikis, Vazquez Velez, Cristina, Žejdl, Petr, Zhang, Yousen, Zogatova, Dominika: The CMS Orbit Builder for the HL-LHC at CERN. EPJ Web of Conf. **295**, 02011 (2024) https://doi.org/10.1051/epjconf/202429502011

[54] Kopeliansky, R.: ATLAS Trigger and Data Acquisition upgrades for the High Luminosity LHC. Technical report, CERN, Geneva (2023). https://cds.cern.ch/record/2871280

[55] Nvidia: OpenSM. [Online]. Available: https://docs.nvidia.com/networking/display/MLNXOFEDv461000/OpenSM. Accessed on: June 24, 2024. (2023)

[56] Gill, P., Jain, N., Nagappan, N.: Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. SIGCOMM Comput. Commun. Rev. **41**(4), 350–361 (2011) https://doi.org/10.1145/2043164.2018477

[57] Yigitbasi, N., Gallet, M., Kondo, D., Iosup, A., Epema, D.: Analysis and modeling of time-correlated failures in large-scale distributed systems. In: 2010 11th IEEE/ACM International Conference on Grid Computing, pp. 65–72. IEEE, Brussels, Belgium (2010). https://doi.org/10.1109/GRID.2010.5697961

[58] Gill, P., Jain, N., Nagappan, N.: Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In: Proceedings of the ACM SIGCOMM 2011 Conference. SIGCOMM '11, pp. 350–361. Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/2018436.2018477

[59] Hensgen, D., Finkel, R., Manber, U.: Two algorithms for barrier synchronization. International Journal of Parallel Programming **17**, 1–17 (1988) https://doi.org/10.1007/BF01379320

[60] Drung, B., Rosenstock, H.: Current OpenSM Routing. [Online]. Available: https://github.com/linux-rdma/opensm/blob/master/doc/current-routing.txt. Accessed on: June 24, 2024. (2017)

[61] Bogdanski, B., Johnsen, B.D., Reinemo, S.-A., Sem-Jacobsen, F.O.: Discovery and Routing of Degraded Fat-Trees. In: 2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 697–702. IEEE, Beijing, China (2012). https://doi.org/10.1109/PDCAT.2012.67

[62] Abdous, S., Sharafzadeh, E., Ghorbani, S.: Burst-tolerant datacenter networks with Vertigo. In: Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies. CoNEXT '21, pp. 1–15. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3485983.3494873 . https://doi.org/10.1145/3485983.3494873

[63] Zhang, Q., Liu, V., Zeng, H., Krishnamurthy, A.: High-resolution measurement of data center microbursts. In: Proceedings of the 2017 Internet Measurement Conference. IMC '17, pp. 78–85. Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3131365.3131375 . https://doi.org/10.1145/3131365.3131375

[64] Shin, J., Pinkston, T.M.: The Performance of Routing Algorithms under Bursty Traffic Loads. In: International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 737–743. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, USA (2003). https://api.semanticscholar.org/CorpusID:17559938

[65] Stein, E., Bramas, Q., Colombo, T., Pelsser, C.: Fault-adaptive Scheduling for Data Acquisition Networks. In: 2023 IEEE 48th Conference on Local Computer Networks (LCN), pp. 1–4. IEEE, Daytona Beach, FL, USA (2023). https://doi.org/10.1109/LCN58197.2023.10223324

[66] Prisacari, B., Rodriguez, G., Minkenberg, C., Hoefler, T.: Fast Pattern-Specific Routing for Fat Tree Networks. ACM Transactions on Architecture and Code Optimization **10**, 1–25 (2013) https://doi.org/10.1145/2541228.2555293

[67] Schweissguth, E., Danielis, P., Timmermann, D., Parzyjegla, H., Mühl, G.: ILP-Based Joint Routing and Scheduling for Time-Triggered Networks. In: Proceedings of the 25th International Conference on Real-Time Networks and Systems. RTNS '17, pp. 8–17. Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3139258.3139289 . https://doi-org.ezproxy.cern.ch/10.1145/3139258.3139289

[68] experts, G.: Gurobi Optimization Inc. Gurobi optimizer reference manual. [Online]. Available: http://www.gurobi.com. Accessed on: June 24, 2024. (2023)

[69] Luppold, A., Oehlert, D., Falk, H.: Evaluating the Performance of Solvers for Integer-Linear Programming. Technical report, Hamburg University of Technology, Hamburg, Germany (November 2018). https://doi.org/10.15480/882.1839

[70] Subramoni, H., Kandalla, K., Jose, J., Tomko, K., Schulz, K., Pekurovsky, D., Panda, D.K.: Designing Topology-Aware Communication Schedules for Alltoall Operations in Large InfiniBand Clusters. In: 2014 43rd International Conference on Parallel Processing, pp. 231–240. IEEE, Minneapolis, MN, USA (2014). https://doi.org/10.1109/ICPP.2014.32

[71] Subramoni, H., Bureddy, D., Kandalla, K., Schulz, K., Barth, B., Perkins, J., Arnold, M., Panda, D.K.: Design of network topology aware scheduling services for large InfiniBand clusters. In: 2013 IEEE International Conference on Cluster Computing (CLUSTER), pp. 1–8. IEEE, Indianapolis, IN, USA (2013). https://doi.org/10.1109/CLUSTER.2013.6702677

[72] Domke, J., Hoefler, T.: Scheduling-Aware Routing for Supercomputers. In: SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 142–153. IEEE, Salt Lake City, UT, USA (2016). https://doi.org/10.1109/SC.2016.12

[73] Rocher-González, J., Gran, E.G., Reinemo, S.-A., Skeie, T., Escudero-Sahuquillo, J., García, P.J., Flor, F.J.Q.: Adaptive Routing in InfiniBand Hardware. In: 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pp. 463–472. IEEE, Taormina, Italy (2022). https://doi.org/10.1109/CCGrid54584.2022.00056

[74] Kasan, H., Kim, G., Yi, Y., Kim, J.: Dynamic Global Adaptive Routing in High-Radix Networks. In: Proceedings of the 49th Annual International Symposium on Computer Architecture. ISCA '22, pp. 771–783. Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3470496.3527389 . https://doi-org.ezproxy.cern.ch/10.1145/3470496.3527389

[75] Rocher-Gonzalez, J., Escudero-Sahuquillo, J., Garcia, P., Quiles, F., Mora, G.: Towards an efficient combination of adaptive routing and queuing schemes in Fat-Tree topologies. Journal of Parallel and Distributed Computing **147** (2020) https://doi.org/10.1016/j.jpdc.2020.07.009

[76] Zahid, F., Gran, E.G., Bogdanski, B., Johnsen, B.D., Skeie, T.: A Weighted Fat-Tree Routing Algorithm for Efficient Load-Balancing in Infini Band Enterprise Clusters. In: 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, pp. 35–42. IEEE, Turku, Finland (2015). https://doi.org/10.1109/PDP.2015.111

[77] Bienkowski, M., Korzeniowski, M., Räcke, H.: A Practical Algorithm for Constructing Oblivious Routing Schemes. In: Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures. SPAA '03, pp. 24–33. Association for Computing Machinery, New York, NY, USA (2003). https://doi.org/10.1145/777412.777418 . https://doi-org.ezproxy.cern.ch/10.1145/777412.777418

[78] Griner, C., Zerwas, J., Blenk, A., Ghobadi, M., Schmid, S., Avin, C.: Cerberus: The Power of Choices in Datacenter Topology Design - A Throughput Perspective. Proc. ACM Meas. Anal. Comput. Syst. **5**(3) (2021) https://doi.org/10.1145/3491050

[79] Zerwas, J., Györgyi, C., Blenk, A., Schmid, S., Avin, C.: Duo: A High-Throughput Reconfigurable Datacenter Network Using Local Routing and Control. Proc. ACM Meas. Anal. Comput. Syst. **7**(1) (2023) https://doi.org/10.1145/3579449

[80] Domke, J., Matsuoka, S., Ivanov, I.R., Tsushima, Y., Yuki, T., Nomura, A., Miura, S., McDonald, N., Floyd, D.L., Dubé, N.: HyperX Topology: First at-Scale Implementation and Comparison to the Fat-Tree. In: Proceedings of the

International Conference for High Performance Computing, Networking, Storage and Analysis. SC '19. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3295500.3356140 . https://doi.org/10.1145/3295500.3356140

[81] Domke, J., Matsuoka, S., Radanov, I., Tsushima, Y., Yuki, T., Nomura, A., Miura, S., McDonald, N., Floyd, D.L., Dubé, N.: The First Supercomputer with HyperX Topology: A Viable Alternative to Fat-Trees? In: 2019 IEEE Symposium on High-Performance Interconnects (HOTI), pp. 1–4. IEEE, Santa Clara, CA, USA (2019). https://doi.org/10.1109/HOTI.2019.00013

[82] Liu, V., Halperin, D., Krishnamurthy, A., Anderson, T.: F10: A Fault-Tolerant Engineered Network. In: 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), pp. 399–412. USENIX Association, Lombard, IL (2013)