# Fault-adaptive Scheduling for Data Acquisition Networks

Eloise Stein
*U. Strasbourg and CERN*, France
eloise.stein@etu.unistra.fr

Quentin Bramas
*U. Strasbourg*, France
bramas@unistra.fr

Tommaso Colombo
*CERN*, France
tommaso.colombo@cern.ch

Cristel Pelsser
*UCLouvain*, Belgium
cristel.pelsser@uclouvain.be

***Abstract*—Supporting such an all-to-all traffic matrix is challenging as it can easily lead to congestion. Scheduling patterns are designed to avoid such congestion by spreading the communications over time. The time is divided in phases and communications are spread across the phases. However, current scheduling algorithms are not fault-tolerant. In this paper we propose a fault-adaptive congestion-free scheduling to support an all-to-all exchange in fat tree topology. Our approach consist in the computation of the minimum number of communication phases required to support the all-to-all exchange with the available links, and of the scheduling of the communications on these phases. It enables to recover from failures and makes optimal use of the remaining bandwidth. We show that our scheduling approach provides better performance than the most common approach which is the Linear-shift scheduling. The throughput is improved by roughly 80% with our approach, for as little as one link failure.**

***Index Terms*—all-to-all, fat-tree networks, integer linear programming**

## I. Introduction and motivation

The personalized all-to-all exchange is a collective communication pattern in which each node sends a different message to every other node participating in the exchange. This pattern, often used by HPC applications, is also needed by data-acquisition (DAQ) applications.

Large-scale scientific instruments are made up of a myriad of independent sensors, each monitoring a small fraction of the same phenomenon. When the instrument observes multiple events over time, each sensor produces a time-ordered stream of data fragments. The DAQ application must then assemble these fragments into coherent snapshots of each observed phenomenon. Each node running the DAQ application is usually responsible for an equal share of the total observed events. These requirements result in a continuous succession of all-to-all exchanges. To be more precise, the data arrives at the servers and then needs to be transmitted to all the other servers using a fat tree topology, as depicted in figure 1. In this topology, the servers are located at the bottom.

To makes best use of the available bandwidth in the network, the full capacity of a link is used for each transmission. If two communication flows use the same link, there is a congestion because the two transmissions need to share the capacity of the link.

Fat-tree networks, such as $k$-ary-$L$-tree, are rearrangeably non-blocking. They are therefore well suited for all-to-all operations in general [1] and DAQ networks in particular [2]. We consider a variant of such networks which is SFT(L;k,...,k) or SFT(L;k,...,k,2k) where L is the number of layers and k is the number of down links connected to each switch at layer L as depicted in figure 2. We consider such topologies to benefit from all the links on the switches at the top layer. More details are provided in section II-C.

Optimal implementations of the all-to-all exchange for fat-tree networks avoid network congestion by dividing the exchange into phases. In each phase every end-node sends one chunk of data and receives another chunk (not necessarily to and from the same other end-node). The communications are assigned to each phase so that the communications are spread over the topology. Existing solutions based on this approach [1] [3] are effective in minimizing the time-to-completion of the exchange, especially when the start of each phase is synchronized [2], but do not degrade gracefully in case of network link failures. With not enough available links, some communications belonging to the same phase are forced to share links, creating congestion and delaying the end of the phase. In the remainder of this paper, we will call this situation a *conflict*.

For example, large-scale DAQ networks at CERN [4] process tens of exabytes per year [5] [6], benefiting 12000 physicists [7]. Such networks uses an all-to-all traffic matrix and a fat-tree topology. Network operators at CERN have recently performed a failure sensitivity analysis on the operational DAQ network of the LHCb experiment. The throughput of the LHCb DAQ application decreases by 10% (least used link failed) up to 50% (most used link failed) with a single link failure [8], highlighting the need of solutions that can handle link failures more gracefully.

In this paper, we design a Fault-adaptive Scheduling solution for fat-tree data-acquisition networks supporting a continuous all-to-all traffic pattern. Our solution makes the best use of the available bandwidth and is robust in the face of link failures. We focus in particular on enabling the system to recover from long-duration failures, i.e. failures that last several hours. This situation is very common in networks [9].

The main contributions of our work are:

- A formula to express the lower bound on the number of phases in case of failures to ensure the communications without congestion.
- An all-to-all scheduling algorithm that accounts for link failures.

- The evaluation of the performance of the solution, especially in case of failures.

## II. Background

In this section, we give the formal definition of logical fat-trees and specialized fat-trees. We also introduce the all-to-all traffic matrix.

### A. All-to-all exchange

All links have the same bandwidth and a flow between a server source and destination completely fills the bandwidth of a link. To avoid congestion, the communications are spread over phases. At each phase, each server sends data to a single destination. Phases are designed such that communications are spread over the topology. Each server is provided with the schedule of the communications for all the phases. Synchronization happens among all servers at the end of each phase such that communications do not overlap, again to prevent congestion. When all servers have sent data to all the other servers, the all-to-all exchange is complete and a new one can begin.

*Definition 1:* An *all-to-all schedule* (or matrix) of length $P$ is a function $\sigma$ that associates each phase $0 \leq p \leq P-1$ with a set of communication flows $\sigma(p) = \{(i,j)\}$ such that $\forall i,j \in S \times S$, where $S$ is the set of servers, $\exists p$, $\sigma(p)(i) = j$. In other words, in the schedule, there exists a phase where server $i$ sends its data to server $j$.

The minimum length of an all-to-all schedule when there is no link failure is obviously $|S|$, the number of servers.

### B. Logical Fat-tree

Logical fat-trees can be described as a traditional tree structure, where the link capacity increases as we move closer to the root. The capacity of a link at any level of the fat tree must be equal to the sum of the capacities of the links at the preceding level [10]. It is common to only consider trees where the number of children is the same for all nodes at a given layer. We use here a definition with notation similar to previous work [3].

*Definition 2:* A logical *fat-tree* $LFT(L; M_1, \dots, M_L)$ is a rooted tree graph of height $L+1$, where the set of nodes is partitioned into $L+1$ sets $V_L$, $V_{L-1}, \dots V_0$. Each set is a layer of the tree. Their size is respectively 1, containing only the root, and then $M_L$ (the nodes are layer L), $M_L M_{L-1}$ (at layer $L-1$), …, $\prod_{i=0}^{L} M_i$ (at layer 0). The link capacity between a node at layer $l$ and a node at layer $l+1$ is $\Pi_l = \prod_{i=1}^{l} M_i$, such as a node has enough bandwidth to serve all its children.

### C. Specialized Fat-tree

Our specialized fat tree, denoted $SFT(L;k,2k)$ corresponds to the biggest fat tree with $L+1$ levels one can build with switches having $2k$ ports. At each level each switch has $k$ parents and $k$ children (like in a $k$-ary-$L$-tree) but the switches at level $L$ have $2k$ children. We call the switches at level $L$ *spine* switches; the switches at level 1 are called *leaf* switches. In the remaining we always consider without loss of generality that each link in a specialized fat tree has a capacity of one.

Our specialized fat tree $SFT(L;k,2k)$ is associated with the logical fat tree $LFT(L;k,\dots,2k)$ and we call a *group* of switches, the set of switches in the specialized fat tree that correspond to a given node in the corresponding fat tree. To avoid confusion, we call *LFT-node* the corresponding node in the logical fat tree associated with a group of switches. A group of switches correspond to a node in the corresponding fat tree.

For example, figure 1 presents a fat tree $LFT(3;2,2,4)$ with the servers numbered from 0 to $2 \times 2 \times 4 - 1 = 15$ and Figure 2 presents the corresponding specialized fat tree $SFT(3;2,4)$ supporting the same number of servers.

### D. Reduced logical Fat-Tree and Fat-tree

It is interesting to consider *reduced logical fat trees* and *reduced fat trees*. A reduced logical fat tree corresponds to a fat tree where the bandwidth available is reduced. A reduced fat tree corresponds to a fat tree where the capacity of some links is reduced. A reduced fat tree can be useful either because some links might not require full capacity, or to simulate a failure. Similarly, a reduced specialized fat tree is obtained from a specialized fat tree by removing some links. However, in this case, removing a link in the corresponding specialized fat tree is not necessarily equivalent to reducing the capacity by one in the logical fat tree. We now define the corresponding link capacity of the logical fat tree associated with a specialized fat tree.

*Definition 3:* Given a reduced specialized fat tree $F$, $l$ a link in the corresponding logical fat tree, and $B$ the set of servers below this link. We say the capacity of $l$ is $c$ if, for any source set $S \subset B$ and destination set $D$ of size $c$, such that $D \cap B = \emptyset$, there exists at least $c$ edge-disjoint paths in the specialized fat tree from servers in $S$ to servers in $D$.

## III. Scheduling with failures

In this section we give a lower bound on the length of an all-to-all schedule in reduced specialized fat-trees. We then propose an algorithm to schedule the communications in the number of phases matching the lower bound.

### A. Lower bound on the number of phases in case of failure

Our lower bound applies to any logical fat-tree $LFT(L; M_1, \dots, M_L)$. We use the same notations as in [3]. Consider that the link capacity between a node at layer $l$ and a node at layer $l+1$ is reduced by $F$, with $1 \leq l \leq L-1$ and $F < \Pi_l$. The minimum number $P$ of phases to support all these communications is at least

$$P \geq \max\left(\Pi_L, \left\lceil \frac{\Pi_l(\Pi_L - \Pi_l)}{\Pi_l - F} \right\rceil\right) \tag{1}$$

The detailed proof of Equation 1 is omitted due to space constraints.

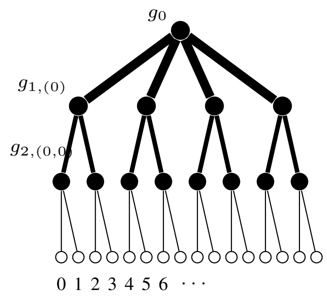

Fig. 1: A logical fat-tree $LFT(3; 2, 2, 4)$ with three layers and two down links at layer one and two, four down links at layer 3.

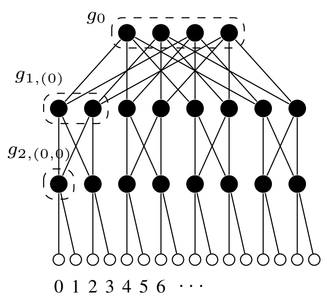

Fig. 2: The associated specialized fat-tree $SFT(3; 2, 4)$. The servers are the white leaves and the switches are filled in black.

### *B. Scheduling the communications on the added phases*

Now that we have the minimum number of phases we need to schedule the communications on these phases. The schedule, determines server interactions in each phase, ensuring no two servers communicate simultaneously. By the end of all phases, every server has communicated with all others.

In this section, we describe our process for computing an all-to-all schedule to enable routing without congestion. The bandwidth-optimal all-to-all pattern [3] proposes a schedule using only half of the full bisection bandwidth. Consequently, the number of used links is more balanced between the different phases compared to linear shift. Although the proposal in [3] is promising, it does not consider the necessary adaptations to the schedule upon link failures. Our proposal introduces an enhanced version of this algorithm that is fault-tolerant and builds a schedule for a reduced fat-tree. To be more precise, the proposal in [3] fits all the communications in $X$ phases, where $X$ is the minimum length of an all-to-all schedule and the number of servers. Upon failures, additional phases need to be added. The minimum number of additional phases is given by the subtraction between the result of equation in 1 and $X$. Our proposal is an algorithm to schedule the communications on these additional phases.

Our algorithm, written in Python, makes sure that the number of communications to and from each switch does not exceed the number of links available on that same switch. The initial schedule results from our implementation of [3]. The communications on links not affected by the failures are left untouched. The algorithm then proceeds in two steps. First, it computes the communications that need to be moved to the extra phases. Next, we employ a Integer Linear Programming (ILP) model using the Gurobi optimizer [11] to distribute communications across the additional phases required in case of link failures. Linear programming systematically tests various solutions to find the most optimized one. However, an optimization function is unnecessary since the minimum number of phases in known thanks to Equation 1.

In our ILP model, binary variables represent communications between server sources and destinations in each phase. These variables determine if there is scheduled traffic between two servers during a specific phase. We generate these variables by iterating through the list of identified communications to be relocated, which was computed in the initial step of the algorithm. The scheduling must respect the following constraints:

- No source server communicates to the same destination server as another source server during the same phase.
- At the end of all phases every server will have communicated to every other server.
- Each source communicates only once in each phase.
- The communications from the leaf switch is less than or equal to its number of available links.
- The communications to the leaf switch is less than or equal to its number of available links.

## IV. Evaluation and Discussion

In this section, we present a comprehensive comparison of our Fault-Adaptive Scheduling (FAS) with other communication patterns. Various all-to-all communication patterns exist, including recursive butterfly, random, and more [12]. However, the most commonly employed communication patterns are XOR and linear shift (LS) [3]. Additionally, we thoroughly evaluate our solution against the bandwidth-optimized exchange pattern (BO) [3], as it holds great promise for improving performance and efficiency compared to linear shift or XOR scheduling patterns.

To simulate the performance of these three scheduling patterns, we use a simple algorithm to compute the number of conflicts that arise at each phase. Since these patterns are algorithmic in nature and the topology is known, the number of conflicts can be easily predicted. In this simulation, we test the schedules on the SFT(2;16,32) topology. First, we determine who talks to whom for each phase based on the selected pattern. Next, by simulating a failure and determining the minimum number of available links on at least one leaf switch, we identify communications that share a link. This is possible because we know which server is connected to the leaf switch with a broken link. In other words, if the maximum number of communications to or from a leaf during a phase is lower than or equal to the number of links available, there is enough bandwidth and so we optimistically assume that there will be no conflicts, so it takes 1 unit of time for the transmission. Otherwise, we assume that the communications are evenly spread on the available links, and that the duration of the phase is determined by the maximum amount of communications that share a link which is computed by this formula: $T = \left\lceil \frac{C}{L} \right\rceil$ where $C$ represents the maximum number of communications requiring the utilization of the links in the given topology, while $L$ denotes the minimum number of links available of the switches. In comparison, since we have no conflicts with our solution, the time it takes to complete a full all-to-all exchange is the result of the formula presented in III.

As mentioned, we consider the topology: SFT(2;16,32). In the figure 3, we show that our solution has better performance than the other three. We define the *slowdown* of a solution in

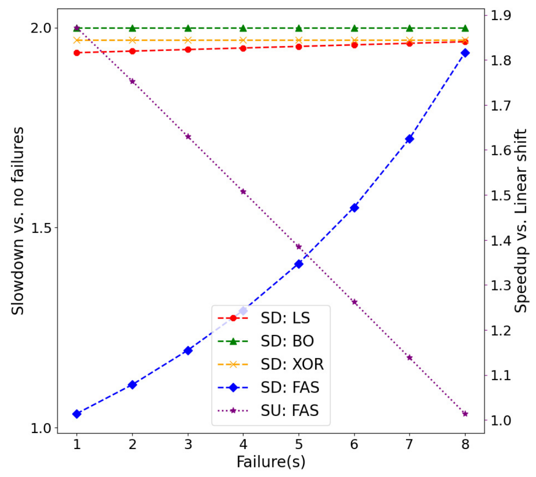

Fig. 3: Time performance with the topology SFT(2;16,32). The left axis shows the slowdown of the different scheduling upon failures compared to no failures, this is the relative time degradation due to failures. The speedup, on the right axis, shows the improvement ratio of FAS compared the linear shift. The failures happen on the leaf switches.

the presence of failures as the ratio between the time for the all-to-all exchange completion with failures and the time to completion with no failures, which is the ideal performance. The time it takes for the all-to-all exchange completion is in the order of milliseconds. However, the unit of time does not matter for the slowdown since it is a ratio. Basically, for every failure scenarios, we compute the slowdown with this simple formula: $Slowdown = \frac{T}{S}$ where $T$ represents the time it takes for the considered all-to-all schedule completion and $S$ is the number of servers which is the minimum length of an all-to-all schedule without failures, as shown in II-A. We define the *speedup* as the improvement ratio of our proposal, FAS, compared to the linear shift. We choose the linear shift because it has the best performance compared to BO and XOR. We compute the speedup with this formula: $Speedup = \frac{T_{LS}}{T_{FAS}}$. The simulation involves eight failure scenarios, which are represented on the x-axis in figure 3. It's important to note that when the number of failures is stated as 1, it does not imply that there is only a single failure in the entire topology. Instead, the result applies when there is one failure on one or more leaf switches, and potentially on all of them. The $x$-axis gives the maximum number of failures on every leaf switch. The lower bound in section III-A remains the same even if each node at layer $l$ has $F$ links failure with their parents at layer $l+1$. In fact, not only the lower bound is the same, but a schedule that tolerates reduction in capacity of $L$ between layer $l$ and $l+1$ also tolerates a reduction of capacity of $F \times M_{l+1}$ between layer $l+1$ and $l+2$.

Our solution results in smaller slowdowns than the other three for all scenarios considered. However, this difference is less pronounced when more than half of the uplinks fail on the same switch. As the number of link failure increases, the amount of available paths decreases, so the possibilities of packing communications in each phase decreases. More importantly, the speedup of our solution compared to the linear shift is roughly about 80% for a single failure while it drops to almost 0% when we have half of the links that failed because we do not have enough paths to optimize the routing.

## V. Related work

Initially, the strategy proposed in paper [3] appeared promising due to its use of only half of the up links at each phase, which suggested easier re-routing in case of failures. However, further investigation revealed that this approach does not effectively address the challenges associated with all-to-all scheduling in the event of failures. In contrast, our solution offers a fault-adaptive scheduling mechanism that guarantees conflict-free paths even in the event of failures. By using an integer linear programming model, our approach ensures that a schedule with a feasible routing solution can be achieved, provided the topology is connected.

## VI. Conclusion and future works

The scheduling described in this paper is able to support link failures in fat tree topology with an all-to-all exchange. It performs better than state-of-the art linear-shift, bandwidth-optimized and XOR scheduling, even with many failures in the network. The next step is to find an appropriate routing algorithm to support the scheduled communications for a data acquisition network.

## References

[1] E. Zahavi, G. Johnson, D. J. Kerbyson and M. K. Lang, "Optimized InfiniBand TM fat-tree routing for shift all-to-all communication patterns," in *ISC*, 2010.

[2] F. Pisani et al., "Design And Commissioning Of The First 32 Tbit/s Event-Builder," *IEEE Trans. Nucl. Sci.*, pp. 1–1, 2023.

[3] B. Prisacari, G. Rodriguez, C. Minkenberg, and T. Hoefler, "Bandwidth-Optimal All-to-All Exchanges in Fat Tree Networks," in *ACM ICS'13*, 2013.

[4] G. Jereczek, G. Lehmann Miotto, and D. Malone, "Analogues between tuning tcp for data acquisition and datacenter networks," in *2015 IEEE ICC*, pp. 6062–6067, 2015.

[5] LHCb Collaboration, "LHCb Trigger and Online Upgrade Technical Design Report," tech. rep., CERN, Geneva, 2014.

[6] LHCb Collaboration, "LHCb Upgrade GPU High Level Trigger Technical Design Report," tech. rep., CERN, Geneva, 2020.

[7] CERN, "Key Facts and Figures – CERN Data Centre." https://information-technology.web.cern.ch/sites/default/files/ CERNDataCentre_KeyInformation_July2019V1.pdf. Online; accessed July 1, 2019.

[8] F. Pisani. personal communication.

[9] P. Gill, N. Jain and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," ACM SIGCOMM'11, p. 350–361, Association for Computing Machinery, 2011.

[10] Leiserson et al., "The Network Architecture of the Connection Machine CM-5 (Extended Abstract)," SPAA'92, p. 272–285, Association for Computing Machinery, 1992.

[11] Gurobi experts, "Gurobi Optimization Inc. Gurobi optimizer reference manual." http://www.gurobi.com, 2023. Online; accessed 4 February.

[12] Timo Schneider, Torsten Hoefler and Andrew Lumsdaine, "ORCS: An Oblivious Routing Congestion Simulator," Tech. Rep. 675, Indiana University, Feb. 2009.