UNIVERSITY OF STRASBOURG

DOCTORAL SCHOOL MSII

MATHEMATIQUES, SCIENCES DE L'INFORMATION ET DE L'INGENIEUR

P H D T H E S I S

to obtain the title of

PhD of Science

of the University of Strasbourg

Specialty : COMPUTER SCIENCE

Defended by

Eloïse STEIN

Smart scheduling and routing for data acquisition networks

prepared at ICUBE, NETWORK Team

defended on October 21, 2024

Jury :

Advisors :	Prof. Cristel PELSSER	-	Université de Strasbourg Université catholique de Louvain (UCLouvain)
	Prof. Thomas NOEL	-	Université de Strasbourg
Reviewers :	Prof. Holger FRÖNING	-	Heidelberg University
	Prof. Piero VICINI	-	National Institute for Nuclear Physics (INFN)
Examiners :	Prof. Géraldine TEXIER	-	École nationale supérieure Mines-Télécom Atlantique (IMT)
	Prof. Stefano SECCI	-	Conservatoire National des Arts et Métiers (Cnam)

Résumé

Le Grand Collisionneur de Hadrons (LHC) est le plus grand et le plus puissant accélérateur de particules au monde. Le long du LHC, quatre détecteurs à particules (ALICE, ATLAS, CMS et LHCb) observent les collisions de particules (également appelées événements) créées par le LHC. Ces détecteurs sont composés de dizaines de milliers de capteurs et d'un système d'acquisition de données (DAQ) dont l'objectif est de collecter les fragments de données de chaque capteur et d'assembler tous les fragments correspondant à chaque événement de collision en un seul ensemble de données. Ce processus, appelé Event Building, implique un échange collectif de type all-to-all entre un ou plusieurs ordinateurs interconnectés en réseau. Cependant, le trafic associé à l'Event Building a tendance à créer des congestions en raison de la nature de l'échange collectif all-to-all, qui nécessite d'utiliser presque toute la bande-passante disponible sur le réseau. Si la congestion n'est pas adressée de manière appropriée, ses effets entravent gravement les performances du système DAQ, ce qui peut entraîner la perte de données expérimentales très précieuses.

Cette thèse présente des approches visant à minimiser la congestion du réseau dans une topologie fat-tree, en utilisant le réseau DAQ du détecteur LHCb (Large Hadron Collider beauty) comme étude de cas. Les réseaux fat-tree sont particulièrement adaptés aux communications à haut débit, comme l'Event Building. Ils ont une bande passante de bisection constante pour toutes les bisections possibles, ils sont aussi réarrangeables et non bloquants. Le réseau DAQ du détecteur LHCb exploite actuellement une combinaison particulière d'un algorithme de routage spécifique aux réseaux fat-tree et d'un algorithme d'ordonnancement du trafic all-to-all optimisé, qui évite complètement la congestion sur le réseau. Cette solution permet des performances élevées dans des conditions nominales, mais une seule défaillance de lien peut réduire considérablement le débit atteint par le réseau DAQ de l'expérience LHCb, le faisant passer d'environ 46 Tbps à environ 30 Tbps.

Nous proposons dans cette thèse une étude statistique des pannes dans les réseaux DAQ. Nous avons pu récolter des données sur les pannes qui sont survenues sur le réseau DAQ de l'expérience LHCb pendant une période de deux mois. Ensuite, nous proposons une évaluation des différents algorithmes de routage spécifiques à InfiniBand et pouvant être utilisés comme solution pour le réseau DAQ de l'expérience LHCb. Enfin, nous avons développé une application permettant de créer un échange all-to-all non synchronisé et nous le comparons à l'échange synchronisé en cas de pannes en effectuant des mesures réelles sur le réseau DAQ de l'expérience LHCb.

Pour répondre à notre problème, nous présentons une étude plus détaillée des différents scénarios de pannes réseau afin de montrer ce qu'ils impliquent en termes de réduction de bande passante et les propriétés qu'une solution d'ordonnancement et de routage devrait respecter pour adresser ces scénarios de pannes. Ainsi, nous proposons FORS (Fault-adaptive Optimized Routing and Scheduling), une solution de routage et d'ordonnancement optimisée en cas de pannes afin de maintenir un débit élevé malgré la réduction de la bande passante.

Abstract

The Large Hadron Collider (*LHC*) is the largest and most powerful particle accelerator in the world. At the LHC, four particle detectors (ALICE, ATLAS, CMS, and LHCb) observe the particle collisions (called events) created by the LHC. These detectors are composed of tens of thousands of individual sensors and a data acquisition (DAQ) system whose objective is to collect fragments of data from each sensor and to assemble all the fragments corresponding to each collision event into a single data set. This process, called *Event Building*, involves an all-to-all collective exchange between one or more network computers. However, the traffic associated with Event Building tends to create congestion because of the nature of the all-to-all collective exchange, which requires using almost all of the available bandwidth in the network. If not mitigated, the effects of congestion severely hamper the performance of the data acquisition system, potentially leading to the loss of valuable experimental data.

This thesis presents approaches to minimize network congestion in a fat tree topology, using the DAQ network of the LHCb (Large Hadron Collider beauty) detector as a case study. Fat-tree networks are particularly suitable for high-throughput communication, like Event Building. They have constant bisection bandwidth for all possible bisections, they are also rearrangeably non-blocking. The DAQ network of the LHCb detector currently uses a specific combination of a routing algorithm specific to fat-tree networks and an optimized all-to-all traffic scheduling algorithm, which completely avoids network congestion. This solution allows for high performance under nominal conditions, but a single link failure can significantly reduce the throughput achieved by the LHCb experiment's DAQ network, dropping it from approximately 46 Tbps to about 30 Tbps.

In this thesis, we propose a statistical study of failures in DAQ networks. We were able to collect data on the failures that occurred in the DAQ network of the LHCb experiment over a two-month period. Then, we present an evaluation of various routing algorithms specific to Infiniband that could be used as a solution for the LHCb experiment's DAQ network. Finally, we developed a non-synchronized all-to-all exchange application, which we compare to the synchronized exchange in the event of failures by conducting real measurements on the LHCb experiment's DAQ network.

To address our problem, we present a more detailed study of different network failure scenarios to demonstrate their impact in terms of bandwidth reduction and the properties that a scheduling and routing solution should have to address these failure scenarios. Thus, we propose FORS (Fault-adaptive Optimized Routing and Scheduling), an optimized routing and scheduling solution designed to maintain high throughput despite bandwidth reduction.

Acknowledgments

As I reach the end of this three-year journey, I think of the many people who have walked this path with me. These years have been filled with challenges and growth, and I could not have made it through without the support, encouragement, and kindness of so many. While it is impossible to fully express the gratitude I feel for everyone who has been part of this experience, I want to take a moment to thank each person who has contributed to this journey.

First of all, I would like to express my heartfelt thanks to my PhD advisor, Cristel Pelsser. Your guidance throughout this research has been invaluable, and it would not have reached this point without your expertise and advice. I truly appreciated our discussions over the past three years, your understanding of this research topic, which was new to both of us, as well as your interest for linear programming and the suggestions that greatly helped me develop the solutions presented in this thesis. Finally, thank you for your incredible support and encouragement, without which this thesis would not have been completed. I think you are an incredible supervisor and any student embarking on this journey, as I did, will be very lucky to have you to guide them.

I would also like to thank the members of my jury who agreed to evaluate my thesis. Thank you to Holger Fröning, Piero Vicini, Géraldine Texier, and Stefano Secci. I sincerely hope that you found my work, which has required a lot of time and effort from myself and all those who contributed, to be both engaging and worthwhile.

Then, many thanks to Quentin Bramas, without whom a significant portion of this work would not have been possible. Your invaluable advice and expertise in mathematics made a major contribution to solving some of the main challenges presented in this thesis. I am deeply grateful for your contributions, which played a crucial role in advancing this research.

I am grateful to Pierre Schaus for providing valuable insights on the optimization challenges I faced during this thesis.

I would also like to thank the entire Online Team at CERN for their contributions to this work and for providing their expertise when needed. In particular, I am grateful to Flavio Pisani and Tommaso Colombo, who dedicated an entire day and night to testing the work presented in this thesis with me. I want to express my thanks to Tommaso Colombo for his supervision throughout this project. My thanks also go to Alberto Perro, Pierfrancesco Cifra and Konstantinos Stavropoulos for the engaging discussions we had over pizza. I appreciate the interesting conversations with Mauricio Feo, whether academic or not, and for his encouragement. Finally, I want to thank Clara Gaspar and Niko Neufeld, the leaders of this team, who perform incredible work every day and supported me, helping me to stay motivated and focused on my future opportunities.

Je remercie Denis, sans qui rien de tout cela ne serait arrivé, ainsi que pour ses conseils, en lui souhaitant bon courage pour la fin de son périple.

Je tiens à remercier Jeanne et Thierry pour leur soutien, leurs conseils et leur présence. Leur force de travail et de caractère m'ont toujours motivée à aller de l'avant.

Je remercie également Adèle, dont la détermination, la joie de vivre et la volonté auront été une source d'inspiration dans mon travail.

Je tiens également à exprimer ma profonde gratitude à mes parents, Yolande et Fabrice, pour leur soutien et leur amour tout au long de ce parcours.

Enfin, et surtout, un grand merci à mon partenaire, Hugo. Il est difficile d'exprimer par des mots à quel point ta présence, ton amour et ton soutien ont compté pour moi tout au long de ce parcours. Tu ne le croiras peut-être pas, mais sans toi, ce travail n'aurait jamais abouti. Tes encouragements et ta gentillesse ont été essentiels pour rendre cette thèse possible. Je n'aurais pas pu terminer ce doctorat sans ton incroyable soutien, et je t'en serai toujours profondément reconnaissante.

This work was mainly funded by the European Council for Nuclear Research (CERN) and partially founded by a grant from the Silicon Valley Foundation (CG1318167).

List of publications

Journals

- Eloïse Stein, Flavio Pisani, Tommaso Colombo and Cristel Pelsser. Measuring Performance Under Failures in the LHCb Data Acquisition Network. IEEE Transactions on Nuclear Science (TNS)
- Eloïse Stein, Quentin Bramas, Flavio Pisani, Tommaso Colombo, Cristel Pelsser. FORS: Fault-adaptive Optimized Routing and Scheduling for DAQ Networks. Computing and Software for Big Science, Springer

Conferences

- Eloise Noelle Stein, Quentin Bramas, Tommaso Colombo, Cristel Pelsser (2023). Fault-adaptive Scheduling for Data Acquisition Networks. The 48th IEEE Conference on Local Computer Networks (LCN). October 2023.
- Eloïse Stein, Flavio Pisani, Tommaso Colombo and Cristel Pelsser. Measuring Performance Under Failures in the LHCb Data Acquisition Network. The 24th IEEE Real Time Conference. April 2024.

Contents

Chapter 1

Introduction

Data acquisition (DAQ) systems play a crucial role in the collection of scientific data [Belyaev *et al.* 2017, Jereczek *et al.* 2015, Bawej *et al.* 2015]. 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 [Borrill *et al.* 2015, Dorelli *et al.* 2022, Update 2011], healthcare [Liu *et al.* 2023, Leung *et al.* 2020] and physics [Belyaev *et al.* 2017, Jereczek *et al.* 2015, Bawej *et al.* 2015]. For instance, DAQ systems at the European Organization for Nuclear Research (CERN)[Jereczek *et al.* 2015, Bawej *et al.* 2015] process tens of exabytes annually[LHCb Collaboration 2014, LHCb Collaboration 2020], significantly contributing to advancements in the field of physics research[CERN 2019]. Such systems are deployed at experiments along the Large Hadron Collider (LHC) at CERN to collect fragmented data from various sensors and assemble them to reconstruct each particle collision event. This process is known as Event Building.

Event building for large collider experiments is typically enabled by a high-throughput and low-latency network of interconnected servers. The data exchanged is produced in real-time by large scientific instruments such as the LHC. The 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, also known as "events". Each sensor is connected to a server which receives its data. To synthesize these data fragments into a unified representation of each event, every server exchanges its data fragments with all others through the network. The resulting network traffic matrix is a continuous succession of all-to-all exchanges. As events are continuously produced in the LHC, 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 thesis.

The all-to-all collective exchange demands significant network resources such as bandwidth. To make best use of the available bandwidth in the network, the full capacity of a link is used for each transmission between servers. If two communication flows use the same link, there is a congestion because the two transmissions need to share the capacity of the link. Consequently, bandwidth utilization in the DAQ network is close to the maximum capacity which makes the all-to-all exchange highly sensitive to failures. As highlighted in the literature [Gill *et al.* 2011], link failures in high-throughput networks, such as those used for collective communications between servers in DAQ networks, are common.

DAQ systems often rely on High Performance Computing (HPC) applications for real-time analysis and efficient processing of significant volumes of data. Optimization of HPC applications is the subject of much research, pushing back the limits of hardware components[Wu *et al.* 2023, Wang *et al.* 2023a], computing architecture[Chirkov & Wentzlaff 2023, Huang *et al.* 2023, Contini *et al.* 2023] and networks[Feng *et al.* 2023]. 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, such as the all-to-all exchange, in HPC applications is extensively discussed and addressed [Prisacari *et al.* 2013b, Prisacari *et al.* 2013a, Al-Fares *et al.* 2010, Izzi & Massini 2020, Zahavi *et al.* 2009, Peng *et al.* 2022, Izzi & Massini 2022, Izzi & Massini 2023], 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 [Singh *et al.* 2021]. As we demonstrate in this thesis, network performance is currently significantly impacted when bandwidth reduction occurs. Addressing this issue is challenging as it involves adapting the scheduling and routing of these collective exchanges to the remaining network bandwidth during failures.

Collective operations, in general, are also increasingly used in machine learning[Sergeev & Del Balso 2018, Zhao *et al.* 2024a, Zhou *et al.* 2023, Wang *et al.* 2023b], including AllGather, AllReduce, or ReduceScatter [Nvidia 2020]. However, they differ from the all-to-all collective exchange. Our problem is specific to DAQ systems.

The all-to-all exchange is a collective communication that is very demanding in bandwidth as all the servers in the DAQ network must exchange data with all the others. If all servers send data to the same destination simultaneously, the links to this destination become congested. A typical strategy to address this problem is to spread the communications to each server over time, meaning that the exchange is segmented into distinct phases.

In this approach, the all-to-all collective exchange is synchronized, meaning that the DAQ application ensures that all servers complete their data exchange before moving to the next phase. This synchronized approach achieves high throughput, particularly in systems with data rates close to 100% of the link capacities [Pisani *et al.* 2023a].

1.0.1 Contributions

In this thesis, I present my contributions to routing and scheduling solutions that improve fault tolerance in DAQ networks. Additionally, I demonstrate, through measurements, the importance of addressing this issue, given the sensitivity of DAQ networks to failures.

First, we propose to study link failures in the DAQ network of the Large Hadron Collider beauty (LHCb) experiment at CERN. We analyze network failures observed over a two-month period during which the LHC was fully active and data were exchanged in the DAQ network. Specifically, we present statistics on the duration, frequency, and underlying causes of network link failures to prove that failures are frequent and can last a long time. These statistics motivate our problem.

Then, we present an analysis of various routing algorithms relevant to the studied network. Specifically, we evaluate the performance of adaptive routing compared to the routing algorithm currently used in the LHCb DAQ network with various failure scenarios. Our measurements highlight the throughput achieved by each routing algorithm on the network under study. We find that link failures can significantly degrade performance, as bandwidth utilization in the DAQ network is close to the maximum capacity. A single failure leads to congestion and degrades the throughput from approximately 46 Tbps to 30 Tbps in total, even with the best routing solution.

We also propose to evaluate the two scheduling approaches of the all-to-all collective exchange: non-synchronized and synchronized. In a synchronized all-to-all exchange, all servers wait for the others to complete their data exchange before proceeding to the next phase. This is the approach currently used in the studied DAQ network. An alternative approach involves a simpler non-synchronized all-to-all exchange, where the network is left to deal with the congestion. These two approaches have never been compared in the context of network link failures, which is what we propose in Chapter 4. In this thesis, we do not consider failures between network switches and servers, as these failures result in complete disconnection of the affected servers from the network. Such servers are no longer able to participate in the all-to-all exchange, and their exclusion does not impact the bandwidth required to complete the exchange among the remaining servers. Finally, we derive some design recommendations from our comparison of the throughput achieved by the synchronized and non-synchronized all-to-all applications in the presence of network failures. However, none of these scheduling and routing approaches enable to make efficient use of the bandwidth that remains in the network when failures occur. These contributions are discussed in Chapter 4 and have resulted in one publication [Stein *et al.* 2024].

As we demonstrate the importance of finding a scheduling and routing solution that ensures more graceful degradation of throughput, we conduct a detailed study of existing scheduling algorithms in the literature. We show that these algorithms are not fault-tolerant, as the number of phases in a synchronized all-to-all exchange is insufficient to prevent congestion. To address this, we introduce the concept of bandwidth reduction, which more precisely defines how failures affect routing in an all-to-all schedule and allows us to derive a formula for computing the lower bound on the number of phases necessary for congestion-free routing, making the number of phases optimal. Following this, we propose an algorithm, along with an Integer Linear Programming (ILP) model, to adapt existing all-to-all scheduling patterns to handle failure scenarios on a fat-tree topology. Finally, we evaluate our contribution and highlight its limitations. This contribution has led to one publication [Stein *et al.* 2023].

To address these limitations, we propose a more comprehensive study of various failure scenarios in the network, focusing on the challenges for scheduling and routing to prevent congestion. Then, we present a Fault-adaptive Optimized Routing and Scheduling (FORS) solution to maintain high all-to-all throughput despite the bottlenecks introduced by network link failures. FORS is composed of 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, combining a route computation algorithm for basic failure scenarios with an Integer Linear Programming (ILP) model designed to address more challenging 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 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. This contribution is under submission at Computing and Software for Big Science in Springer journal.

1.0.2 Overview

This thesis is composed of seven chapters. Chapter 2 provides the necessary background to understand our contributions. We start by introducing the fat-tree topology and the two variants we consider. Then, we discuss the all-to-all collective exchange and the linear-shift scheduling algorithm, which is widely used in InfiniBand networks. We then provide an overview of the DAQ network of the LHCb experiment and the Event Building process. We present with more details the application of the all-to-all collective exchange in the Event Building process and the use of InfiniBand technology in the fat-tree topology of the DAQ network.

In Chapter 3, we review related work relevant to our research, including various existing collective operations other than all-to-all. We also present different routing strategies, such as adaptive, oblivious, and scheduling-aware. Finally, we introduce other network topologies such as Dragonfly, which is widely used in HPC networks, along with demand-aware topologies. We introduce the HyperX and F10 topologies that are relevant to our problem as they could improve fault tolerance by increasing the number and diversity of paths available between switches.

In Chapter 4, we present our statistics on network failures that occurred in the studied network during March and April 2024. We then present measurements of the throughput achieved by different InfiniBand routing algorithms relevant to our network, along with data showing the effects of the synchronization of the all-to-all exchange in terms of scalability and in the event of failures. We conclude this chapter with design recommendations.

In Chapter 5, we study with more details various scheduling algorithms and demonstrate their inadequacy in the event of failures due to inevitable congestion, as no congestion-free routing solution exists with the proposed number of phases. We then explain how we compute the optimal number of phases based on the bandwidth reduction caused by network failures. Finally, we propose a first scheduling solution that can adapt any all-to-all scheduling pattern in the event of failures, addressing simple failure scenarios only.

In Chapter 6, we conduct a more in-depth analysis of different failure scenarios and their consequences in terms of bandwidth reduction. We then propose a scheduling algorithm that more effectively consider routing, ensuring that a feasible routing solution exists without congestion for any failure scenario. Furthermore, we present a routing algorithm associated with an Integer Linear Programming (ILP) model to address all possible failure scenarios as long as the network switches directly connected to the servers remain fully connected. Finally, we evaluate the performance and applicability of our solution.

In Chapter 7, we conclude by summarizing our various contributions and their results, as well as suggesting further research perspectives.

Have a pleasant reading.

Chapter 2

Background

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

2.1 The Fat-tree topology

Our contributions currently focus on a single topology family: Generalized Fat-trees, more specifically k-ary-L-trees [Petrini & Vanneschi 1997]. Fat-tree topologies are widely used in High-Performance Computing (HPC). Many of the systems on the latest Top500 list employ a fat-tree topology [List 2024]. Generalized fat-tree topologies are particularly well-suited for the all-to-all collective exchange, thanks to their scalability and rearrangeable non-blocking nature[Pippenger 1978]. This last characteristic allows each end node to communicate at the same time, making the best use of the available bandwidth [Yao *et al.* 2014]. Fat-trees also maintain constant bisection bandwidth across all possible bisections. A bisection is when the network is divided into two equal halves. Bisection bandwidth refers to the total bandwidth available across the set of links that, when cut, would split the network into these two equal halves. In this section, we define a logical fat-tree and the generalized fat-trees we consider.

2.1.1 Terminology

For simplicity, in this thesis, we use the term *server* to refer to network end-nodes, and *switch* for intermediate network nodes. The difference between the two in the problem addressed in this thesis is that the server processes the data, while the switch simply forwards it to its destination.

2.1.2 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 [Leiserson et al. 1992]. It is common to only consider trees where the number of children is the same for all switches at a given layer. The main property of a fat-tree is that any communication flows is possible without congestion, *i.e.*, it is non-blocking. In more details, for any bijection f between the set of servers then it is possible for each server i, at the same time, to communicate with the server

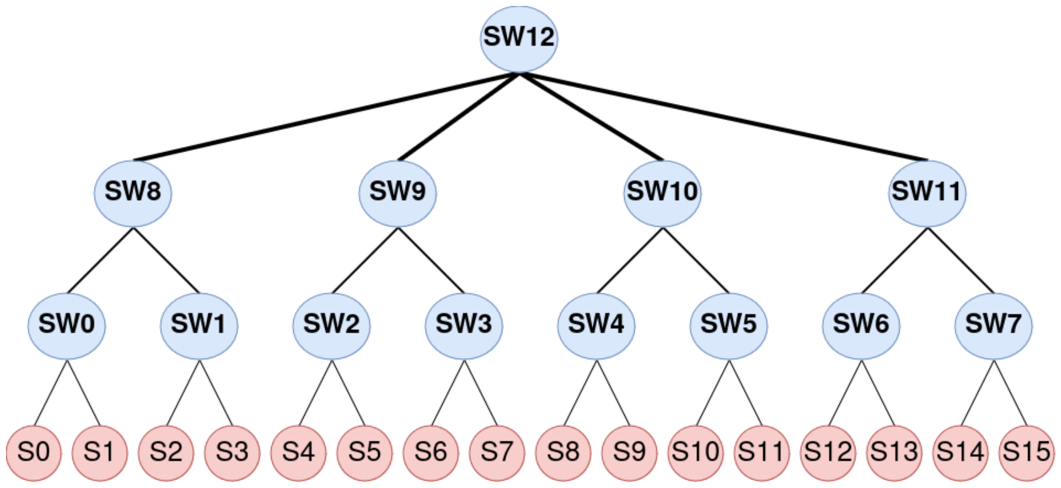


Figure 2.1: Logical fat-tree $LFT(3;2,2,4)$ with three layers. At the top layer, the switch $SW12$ has 4 down links with a capacity of 4. Then, each switch at layer 2 $SW8..11$ and each switch at layer 1 $SW0..7$ has 2 down links with a capacity of respectively 2 and 1. There are 16 servers connected in the topology. The nodes $SW0 \dots 12$ are switches and the nodes $S0 \dots 15$ are servers.

$f(i)$ using all the available bandwidth without congestion. We use here a definition with notation similar to previous work [Prisacari *et al.* 2013b].

Definition 1 *A logical* fat-tree $LFT(L;M_1,\dots,M_L)$ *is a rooted tree graph of height* $L+1$*, where the set of switches 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 at the top layer of the topology, and then* M_L *(the number of switches at layer* L*),* $M_L \times M_{L-1}$ *(at layer* $L-1$*),* $\dots$*,* $\prod_{i=0}^{L} M_i$ *(at layer 0). The link capacity between a switch at layer* l *and a switch at layer* $l+1$ *is* $\Pi_l = \prod_{i=1}^{l} M_i$*, such as a switch has enough bandwidth to serve all its servers.*

The servers from the set Servers of nodes are denoted server *nodes, and represent the only nodes that can send and receive messages. The other nodes in the tree are here to route messages and are switches.*

An example of a logical fat-tree is depicted in Figure 2.1 which presents the fat-tree $LFT(3;2,2,4)$ with the servers numbered from 0 to $2 \times 2 \times 4 - 1 = 15$.

2.1.3 The considered Generalized Fat-trees

The practical scalability of fat-tree topologies is constrained by the increasing link capacities required as we approach the root. However, regular fat-tree topologies, where each node has the same number of children at every level, can be generalized into a scalable variant using readily available hardware. This is achieved by replacing the higher-bandwidth links with multiple parallel links of equal bandwidth, and by replacing each node with a set of switches that have the same number of ports.

Our generalized fat-tree, denoted $FT(L; k, \ldots, 2k)$ corresponds to the biggest fat-tree with $L+1$ levels one can build with switches having up to $2k$ ports. We use here a definition with notation similar to previous work [Prisacari *et al.* 2013b]. 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 generalized fat-tree has a capacity of one.

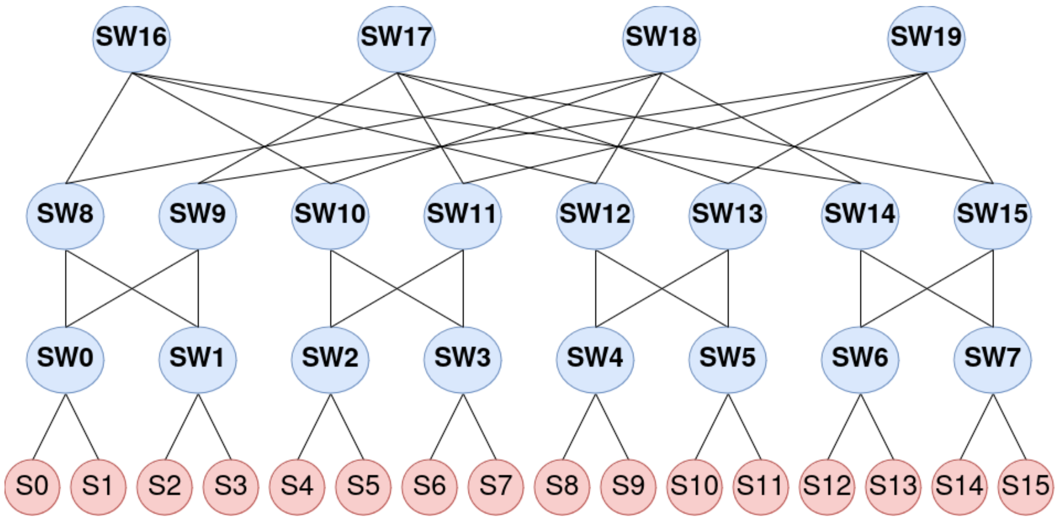


Figure 2.2: Generalized fat-tree $FT(3; 2, 2, 4)$ with three layers. At the top layer, each switch $SW16..19$ has 4 down links. Then, each switch at layer 2 $SW8..15$ and each switch at layer 1 $SW0..7$ has 2 down links. There are 16 servers connected in the topology. Each link has a capacity of 1.

For instance, Figure 2.2 presents a fat-tree $FT(3; 2, 2, 4)$ with the servers numbered from 0 to $2 \times 2 \times 4 - 1 = 15$. The fat-tree $FT(3; 2, 2, 4)$ is the generalized version of the logical fat-tree $LFT(3; 2, 2, 4)$ depicted in Figure 2.1.

The most well-known generalizations are the k-ary-L-tree [Petrini & Vanneschi 1997] that generalizes the fat-tree $FT(L, k, \ldots, k)$ and some variants such as the mirrored-k-ary-L-tree used in [Li & Chu 2019]. The k-ary-L-tree is a tree structure made up of switches, where each switch has an equal number of uplinks and downlinks [Leiserson 1985, Petrini & Vanneschi 1997]. An example of the generalized fat-tree $FT(L, k, \ldots, k)$ is depicted in Figure 2.3. The difference between the k-ary-L-tree generalization of a fat-tree and the generalization $FT(L; k, \ldots, 2k)$ we define is that instead of using only half of the ports on the switches at the highest level in the hierarchy, we use all links to connect the layer below.

In the following of this thesis, when we use the term *generalized fat-tree*, we include the k-ary-L-tree generalization $FT(L; k, \ldots, k)$ and our generalization $FT(L; k, \ldots, 2k)$.

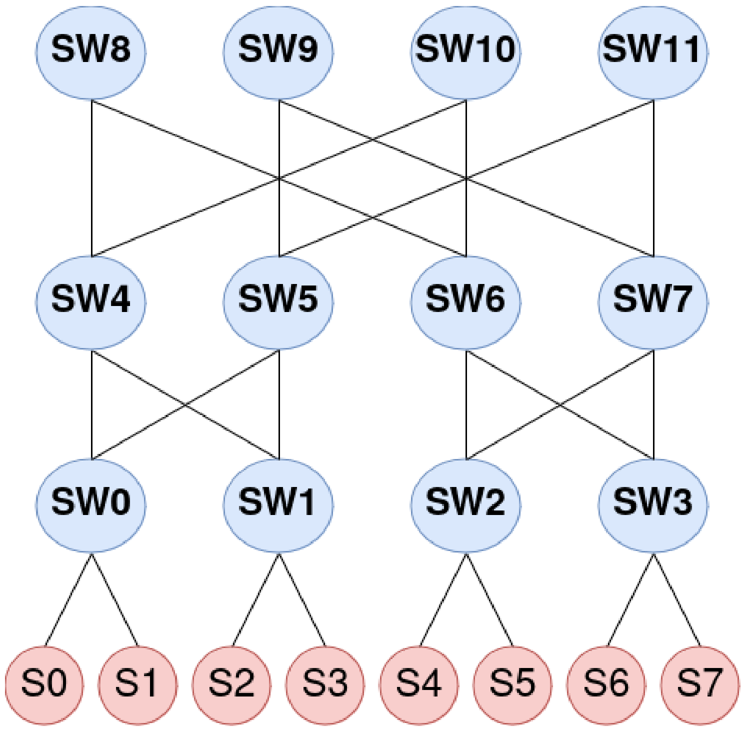


Figure 2.3: Generalized fat-tree $FT(3; 2, 2, 2)$ with three layers. At the each layer, each switch has two down links. There are 8 servers connected in the topology. Each link has a capacity of 1.

2.1.4 Reduced Logical Fat-Tree and Generalized Fat-tree

One of our contributions is the definition of the *reduced logical fat-trees* and the *reduced generalized fat-trees*. A reduced logical fat-tree corresponds to a fat-tree where the bandwidth available is reduced, *i.e.*, the capacity of some links is reduced. A reduced generalized fat-tree also corresponds to a fat-tree where the available bandwidth is reduced; however, in this case, some links completely disappear from the topology, as all links in the generalized fat-tree have a capacity of one. A reduced logical or generalized fat-tree can be useful either because some links might not require full capacity in a logical fat-tree, or to simulate a failure.

2.1.5 Conflicts

A *conflict* is a congestion that happens due to sharing of a link by multiple communication flows. A conflict occurs when a link is used by two different source nodes during the same phase and in the same direction. Assuming that servers always send at full link capacity, the notion of conflict and congestion are equivalent.

In this thesis, we propose a routing and scheduling solution to route the scheduled communication between servers on a reduced fat-tree topology while ensuring that no conflicts occur on the links. Our solution is applicable to both logical and generalized fat-trees. However, this thesis focuses on generalized fat-trees, specifically $FTL; k, \dots, 2k$ and $FTL; k, \dots, k$, as they are more commonly used in practice due to the availability of network hardware.

2.2 The all-to-all collective exchange

The all-to-all collective exchange finds common application in parallel or distributed computing environments and is standardized in Message Passing Interface (MPI) [MPI 2021]. This data exchange plays a crucial role in various scientific applications, such as the Fast Fourier Transform (FFT) algorithm [Dalcin *et al.* 2019, Czechowski *et al.* 2012, Doi & Negishi 2010, Namugwanya *et al.* 2023], which finds application in diverse scientific fields, including healthcare with medical image reconstruction in Magnetic Resonance Imaging (MRI) [Sumanaweera & Liu 2005]. Collective all-to-all communications are also widely used in scientific applications, including other CERN experiments [Jereczek *et al.* 2015, Bawej *et al.* 2015] and large-scale high-performance computing (HPC) applications in general [Chan *et al.* 2007]. Furthermore, the all-to-all exchange is widely used in Grid-based Simulations, employed in the prediction of weather patterns [Müller *et al.* 2019] or in Hadoop-like applications [Roy *et al.* 2015]. More recently, collective exchanges have also been integrated into machine learning applications [Zhao *et al.* 2024b, Shah *et al.* 2023, Cai *et al.* 2021]

2.2.1 Definition

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. A *communication flow* between a source server and a destination server is defined as follow:

Definition 2 *A* communication flow $\boldsymbol{x}(S, P)$ *is a bijective function that associate a server in* S *at a phase in* P *to a server in* D*.*

2.2.2 Segmentation of the all-to-all exchange into phases to prevent congestion

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 P. This implies that when a server completes its exchange with another server at phase $p_0 \in P$, it must wait until all other servers finish their exchange for phase p_0 before communicating with the

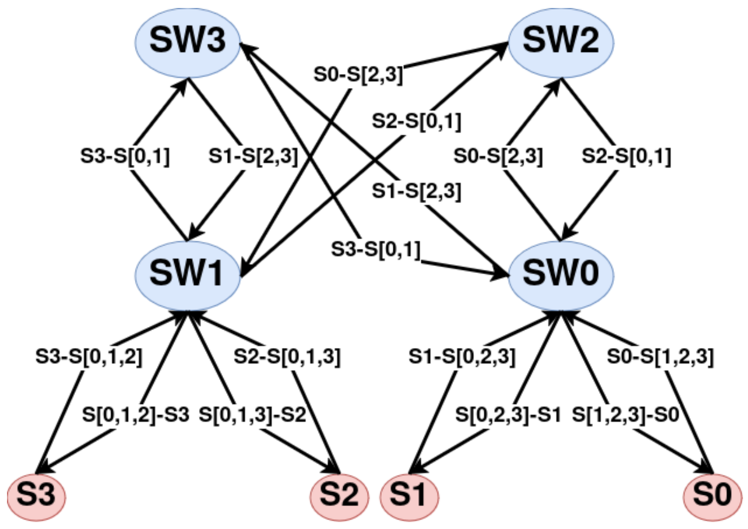


Figure 2.4: A collective all-to-all exchange between 4 servers on a fat-tree topology FT(2;2,2). The switches are $SW[0,\ldots,3]$ and the servers are $S[0,\ldots,3]$. Each server transmits simultaneously to all other servers, creating congestion on all links in the network.

server scheduled for phase $p_1 \in P$. To synchronize the exchange, Barrier algorithms are typically used [Hensgen *et al.* 1988]. Each server communicates with exactly one server at each phase. After the end of all phases, every server has exchanged data with all others, needing only $|S|$ bandwidth units per phase.

Figure 2.4 illustrates an all-to-all exchange between four servers in the fat-tree topology FT(2;2,2), where each server sends data to all others. This results in congestion across all links: the links between the servers and switches $SW[0,1]$ carry three communication flows each and the links between switches $SW[0,1]$ and $SW[2,3]$ handle two flows each in the same direction. To prevent congestion, a more effective strategy is to divide the data exchange into four distinct phases.

In more detail, the all-to-all schedule divided into phases can be defined as follows:

Definition 3 *An* all-to-all schedule *(or matrix) of length P is a function σ 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, to avoid congestion in the network.

2.2.3 Linear-shift scheduling algorithm

To avoid congestion, the all-to-all exchange typically needs to be divided into phases, with the communications between servers being scheduled at each phase. To com-

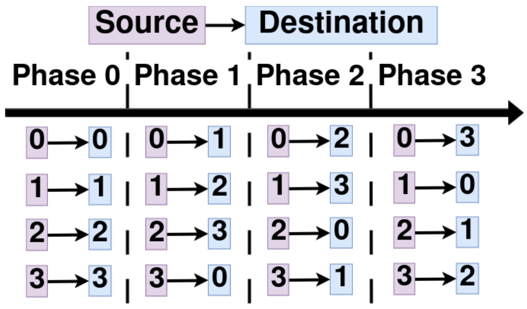


Figure 2.5: The linear-shift scheduling with four servers. Each process communicates with all other servers. To prevent congestion, the exchange is divided into four phases. At each phase, each server sends data to another process and receives data from another one. The minimum number of phases for the all-to-all exchange is the same as the number of servers.

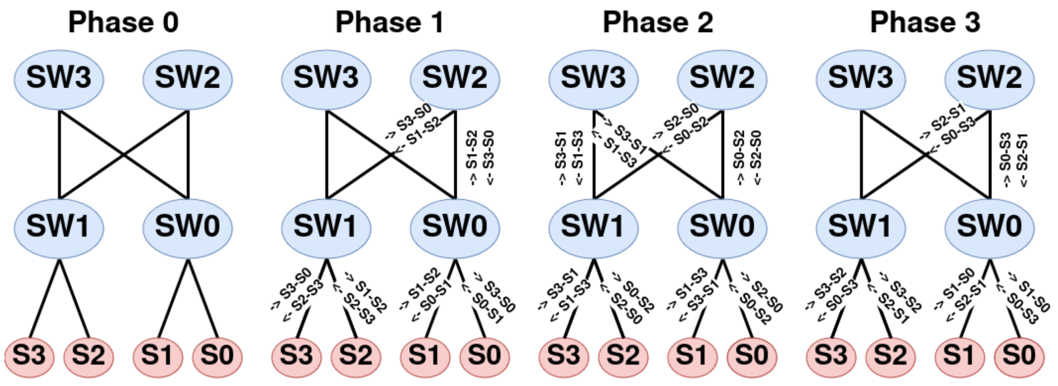


Figure 2.6: The routing of the linear-shift all-to-all schedule with four servers in the fat-tree topology $FT(2; 2, 2)$. No links are used twice in the same direction.

plete 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 [Zahavi *et al.* 2009] illustrated in Figure 2.5, where each destination is computed with $d = (s + p) \bmod S$, where s refers to the source server, p is the phase and S is the total number of servers. Many other scheduling algorithms [Izzi & Massini 2022] exist, such as XOR or Bandwidth-optimal [Prisacari *et al.* 2013b]. 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.

Figure 2.6 shows the routing of the linear-shift all-to-all schedule with four servers in the fat-tree topology $FT(2; 2, 2)$. Each link carries at most one flow in each direction, ensuring that no congestion occurs on the network at any phase.

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 [Müller *et al.* 2019], including data acquisition (DAQ) in particle physics experiments [Jereczek 2017, Amoiridis *et al.* 2023, Pisani *et al.* 2023b] at the European Council for Nuclear Research (CERN). Our case of study is the DAQ network of the LHCb experiment at CERN depicted in Figure 2.7.

2.3.1 The Event Builder network

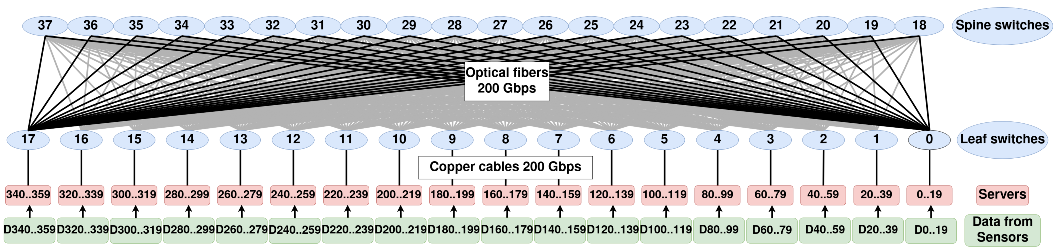


Figure 2.7: The Event Builder Network of the LHCb experiment. The data fragments from the sensors are transmitted to the servers, which perform an all-to-all exchange to reconstruct the events. The network topology is a two-layer fat-tree with 28 40-port InfiniBand High Data Rate (HDR) switches. Each leaf switch is directly connected to approximately 20 servers and 20 spine switches with 200 Gbps links. Each source server traverses the fat-tree topology from the leaf switch it is connected to, up to the spine switches, and then back down to the leaf switches to reach the destination server.

The Large Hadron Collider (LHC) at CERN is the largest and most powerful particle accelerator in the world. Within this accelerator, two particle beams are accelerated to nearly the speed of light before colliding. After over three years of upgrades and maintenance, the LHC currently achieves a record energy level of 13.6 TeV. The resulting physical phenomena from these collisions are studied by physicists, significantly advancing the field of physics research [CERN 2024]. Throughout the remainder of this thesis, a physical phenomenon is referred to as **an event**. 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. Data acquisition (DAQ) systems play a crucial role in collecting data from these sensors and processing it to extract relevant information for physics research. 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 interconnected servers. The Event Building process is our primary focus in this thesis. 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.

In the DAQ system of the Large Hadron Collider beauty (LHCb) experiment, such network is known as the Event Builder network and is illustrated in Figure 2.7. After the Event Building process, interesting events are selected through a two-step High Level Trigger (HLT) process [Aaij *et al.* 2020].

Unlike other experiments such as ATLAS [Aad *et al.* 2023], the LHCb experiment has no hardware trigger, meaning that events are not filtered prior to the event building stage. Instead, LHCb adopts a triggerless DAQ framework, requiring the system, including the communication network, to handle the full collision rate. This results in higher bandwidth requirements. For this reason, the performance of the DAQ network at LHCb is crucial, as it needs to sustain a high data rate of approximately 46 Tbps. Such performance makes the studied network the one with the highest real-time data bandwidth compared to other experiments at CERN [Amoiridis, Vassileios *et al.* 2024, Kopeliansky 2023].

2.3.2 The all-to-all exchange applied to the Event Builder network

In the Event Builder network, the servers receive a partial spacial view of the activity in the beam from a subset of the sensors. Then, the servers send the data from different time intervals to different servers. Each server reconstructs from the received data fragments a complete view of the collision events for its allocated times and is tasked with handling an equal share of the total number of events. In the Event Builder network, each link are used at full capacity by communication flows. This data distribution and reconstruction process results in a continuous succession of all-to-all exchanges. The nature of the all-to-all exchange poses significant challenges from a network perspective. It demands substantial bandwidth to prevent congestion. More precisely, a data fragment denoted as $D(s_i, t)$ from an event occurring at time $t \in T$, is collected by a sensor, and then transmitted to a designated server, denoted as s_i, $i \in [0, S]$, with S being the set of servers IDs in the DAQ application. T defines the set of all possible timestamps at which events occur. To assemble the data fragments, they need to be assigned to a specific server, s_j, that receives all the pieces of data for time $t \in T$ through the DAQ application A as shown in Equation 2.1.

$$\begin{aligned} A : (S \times T)^{|S|} \to S : \\ A(D(s_0, t), D(s_1, t), \ldots, D(s_S, t)) = s_j \end{aligned} \tag{2.1}$$

2.3.3 The routing of the communications

The studied DAQ network relies on the Infiniband technology. Infiniband is widely used in HPC[List 2024]. The fat-tree topology of the studied DAQ network, illustrated in Figure 2.7, is composed of 28 40-ports Infiniband High Data Rate (HDR) switches. As a reminder of Section 2.1, 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 **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 used 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 4.3.3 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 [Nvidia 2023b]. The controller is responsible for computing routes and pushing them into the Linear-Forwarding Tables (LFT) of the switches [Nvidia 2023b]. To efficiently route each communication flow, a routing algorithm compatible with the linear-shift scheduling [Zahavi *et al.* 2009] 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 use a distinct spine for each communication flow.

A routing algorithm with those characteristics is provided by Infiniband. This routing algorithm, named Ftree [Nvidia 2023b], uniformly distributes the traffic on the links between spine and leaf switches according to the destination server. The Ftree routing algorithm assigns ports to join each server in the routing table of the switches. For each destination server, the algorithm starts at the leaf switch directly connected to that server and assigns the output port that leads to it. This process is then repeated for each of the other leaf switches and subsequently for the switches in the upper layers. This step goes from the bottom layer of the network to the top, visiting each leaf switch and assigning the output port that directs traffic toward the destination server. This ensures that every switch has a route to the destination servers. Each switch port has a counter that increments every time the port is used as an output to reach a destination server. If multiple ports are candidates for the same destination, the algorithm selects the port with the smallest port ID [Zahavi *et al.* 2009].

This type of routing algorithm has demonstrated its effectiveness in routing shift communication patterns, such as the linear-shift pattern, without causing congestion on fat-tree topologies commonly used in practice [Jacobs 2010]. 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 [Zahavi *et al.* 2009].

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 [Nvidia 2023b].

2.4 Conclusion

In this chapter, we discussed the background necessary to understand the contributions presented in this thesis. We presented the fat-tree topology and its variants, which is ideal for all-to-all communication due to its non-blocking nature and the bandwidth it offers. Specifically, we discussed the generalization of fat-tree topologies and defined another generalization commonly used in practice, to which all the solutions proposed in this thesis apply.

We then discussed the all-to-all exchange and the strategies implemented to avoid congestion in this collective operation, which requires significant bandwidth. We introduced the linear-shift algorithm, which schedules all-to-all communications without creating congestion.

Finally, we presented our case of study: a data acquisition network in production at CERN. We detailed the activity of the LHCb experiment where this data acquisition network is used. Finally, we showed that the all-to-all exchange, when combined with a fat-tree topology and the Infiniband Ftree algorithm, achieves optimal performance, enabling the data acquisition network to reach a throughput of 46 Tbps when the are no link failures.

Chapter 3

Related Work

Contents

In the previous chapter, we provided a comprehensive overview of the important concepts necessary for understanding our contributions. In this chapter, we discuss the related work in this area. We describe various collective operations other than the all-to-all exchange, which was discussed in the previous chapter, and explain why the All-to-All operation is particularly challenging to optimize in terms of bandwidth compared to other collective operations.

Then, we explore various routing strategies that can be applied to the all-to-all exchange in a fat-tree topology. However, these strategies may not necessarily be the most effective for traffic in the data acquisition network we are studying. By discussing these alternatives, we aim to provide a comprehensive understanding of their strengths and limitations in different contexts.

Finally, we present the different topologies commonly used in data center networks that could be considered for all-to-all communication patterns. We discuss their applicability to our problem, considering how these topologies can impact performance and optimization strategies in the context of data acquisition networks.

3.1 Collective operations

Collective operations involve communication and data sharing between a set of processes P. The most common collective operations are standardized in Message Passing Interface (MPI) [MPI 2023, Nvidia 2020] and are described below:

Broadcast: The Broadcast operation is used when a single process needs to share data with all other processes. For example, in a set of processes $P = \{p_1, p_2, p_3\}$, process p_1 needs to send data d to all other processes. In the Broadcast exchange, process p_1 sends data d to p_2 and p_3, as well as itself. By the end of the exchange, all processes p_1, p_2, and p_3 have received the data d. This operation involves one sender (p_1), $|P|$ receivers (p_1, p_2, p_3), and requires a bandwidth of $|P|$, as only one process sends $|P|$ messages.

Reduce: The Reduce operation aggregates data from all processes into a single result using a specified operation, such as the sum of the maximum of values. Each process contributes its data, and these contributions are combined. The final result is then stored on a designated process. For example, a process $p_1 \in P$ needs to receive the sum of the data d_1 from p_1, d_2 from p_2, and d_3 from p_3. In a Reduce exchange, the sum, or any other operation, is computed progressively between the processes in a hierarchical manner. In our example, it means that instead of each process $p_x \in P$ with $x \in \{1, 2, 3\}$ sends data d_x directly to process p_1 to sum the data, each process sends data d_x to another process. The receiving process sums the value it receives with its own value and forwards the result to the next process and so on until p_1 receives the final sum and add its own value. This operation involves $|P|$ senders (p_1, p_2, p_3), one final receiver (p_1), and requires a bandwidth of $|P|$, as $|P|$ processes each send one message to one process.

All-Reduce: The All-Reduce operation extends the Reduce operation by not only aggregating the data but also ensuring that every process receives the final result. After the reduction operation, the result is broadcast back to all processes. This operation involves $|P|$ senders (p_1, p_2, p_3), $|P|$ receivers (p_1, p_2, p_3), and requires a bandwidth of $|2P|$, as $|P|$ processes each send one message to one process and then the result is sent to $|P|$ processes.

Gather: In the Gather operation, data from all processes is collected and sent to a single process. For instance, process $p_1 \in P$ needs to receive the data d_1 from p_1, d_2 from p_2, and d_3 from p_3. In a Gather exchange, each process $p_x \in P$ with $x \in \{1, 2, 3\}$ sends data d_x to process p_1. This operation involves $|P|$ senders (p_1, p_2, p_3), one receiver (p_1), and requires a bandwidth of $|P|$, as each of the $|P|$ processes sends one message to a single process, the process that receives the result then broadcasts it to all $|P|$ processes.

All-Gather: The All-Gather operation is an extension of Gather, where instead of just one process collecting data from all others, every process collects data from every other process. After the operation, each process has a complete set of data. In an All-Gather exchange, each process $p_x \in P$ with $x \in \{1, 2, 3\}$ sends data d_x to processes $p_{1,2,3} \in P$. This operation involves $|P|$ senders (p_1, p_2, p_3), $|P|$ receivers (p_1, p_2, p_3), and requires a bandwidth of $|P|^2$, as $|P|$ processes each send one message to $|P|$ processes.

Scatter: The Scatter operation is essentially the opposite of Gather. Here, a single process sends a set of data to all processes. Specifically, the data is divided into chunks, and each chunk is sent to a different process. In a Scatter exchange, process $p_1 \in P$ has data d, which is divided into three chunks of data d_1, d_2, d_3.

Process p_1 sends data d_1 to p_1, data d_2 to p_2, and data d_3 to p_3. This operation involves one sender (p_1), $|P|$ receivers (p_1, p_2, p_3), and requires a bandwidth of $|P|$, as one process sends $|P|$ data.

As discussed in Section 2.2, the all-to-all collective exchange is a communication pattern where each process exchanges data with every other process. The bandwidth required for the all-to-all exchange is $|P|^2$. Consequently, the bandwidth required for the all-to-all and all-gather collective exchanges is the same. However, the number of different messages exchanged differs. For all-gather, the number of different messages exchanged is $|P|$ as each process sent a same message to all processes, whereas for all-to-all, the number of different messages exchanged is $|P|^2$, since each process sends a different message to all processes.

To elaborate, in the all-to-all exchange involving processes $p_1, p_2, p_3 \in P$, process p_1 holds a data set $\{a_1, a_2, a_3\}$ and transmits data a_1 to p_1, a_2 to p_2, and a_3 to p_3. As explained in Section 2.3.1, each server connected to the LHCb experiment's Event Builder network receives data fragments from the sensors, and each fragment must be transmitted to an assigned process. This difference between all-to-all and all-gather makes scheduling for all-to-all more challenging to optimize in the event of failures. In the all-to-all exchange, it is not sufficient to simply broadcast the same data to other processes as each data fragment to be transmitted is unique and held by a specific process. This is not the case for all-gather, which can be optimized as shown, for example, in the work by Liangyu Zhao et al. [Zhao *et al.* 2024b], which introduces ForestColl, a tool that generates a forest of spanning trees for aggregation (all-reduce) and broadcasting (all-gather) on any topology while minimizing network congestion.

Optimizing the all-to-all exchange is a challenging problem, especially in the event of failures, because this exchange is highly demanding in bandwidth. Any reduction in bandwidth can lead to congestion if the scheduling is not optimized to consider the new bandwidth available. Numerous solutions have introduced scheduling algorithms that optimize bisection bandwidth [Prisacari *et al.* 2013b, Al-Fares *et al.* 2010, Bruck *et al.* 1997] or are applied to diverse network topologies such as Butterfly [Izzi & Massini 2020, Izzi & Massini 2023] and Dragonfly [Prisacari *et al.* 2013a]. However, these approaches do not consider failure scenarios which is the subject of this thesis.

The all-to-all traffic matrix can also be used by Network on Chip (NoC) [Chatti *et al.* 2010, Venkataramani *et al.* 2022]. A Network on Chip (NoC) is a network on an integrated circuit that connects components within a System on Chip (SoC). The most commonly used topologies for NoCs are mesh and torus, or tree topologies such as fat-trees [Bokhari & Parameswaran 2016]. However, in most NoCs, the traffic pattern is not always all-to-all but tends to be more localized on specific nodes, depending on the application's requirements. Furthermore, scheduling strategies are often used to optimize the network in terms of energy consumption [Tariq *et al.* 2021, Huseyin & İmre 2018, Yao *et al.* 2022] and latency [Chao *et al.* 2012], but not necessarily in fault-tolerance. NoCs are used in a variety of real-time applications, such as multimedia processing. For example, in video processing, NoCs enable different processors to collaborate in real-time to decode, filter, and compress videos [Peng *et al.* 2023].

3.2 Routing strategies

In this section, we discuss various appropriate routing strategies to optimize the all-to-all exchange, with a focus on scenarios involving network failures and efficient bandwidth utilization. We delve into how different approaches can be employed to maintain high performance and reliability, even under challenging conditions such as unexpected network failures.

3.2.1 Scheduling-Aware Routing

Scheduling-aware routing is a network routing strategy that considers not just the best path for routing data but also the scheduling of communications. In this approach, routing decisions are made based on both the current state of the network and the schedule of communication flows, ensuring that resources like bandwidth are efficiently allocated among the communications. For instance, in case of network failures, the network does not just find the shortest or least congested path for data. Instead, it considers when and how network resources, such as bandwidth, will be available. This helps avoid conflicts and congestion in the network.

Various Scheduling-Aware Routing solutions exist [Huang *et al.* 2020, Munir *et al.* 2016], especially for routing the all-to-all collective exchange [Subramoni *et al.* 2014, Subramoni *et al.* 2013, Domke & Hoefler 2016]. In [Subramoni *et al.* 2014], the authors attempted to make the all-to-all communication schedule free of congestion by spreading the traffic more evenly over time, *i.e.*, by adding more phases to perform an all-to-all exchange. They are able to limit the number of communications sharing a link in the all-to-all schedule to two. This strategy is interesting and we propose in the Chapter 5 of this thesis an extension of it by defining a lower bound on the number of phases to perform an all-to-all exchange without any congestion on the links, *i.e.*, no communication flows share a link in the same direction, based on the remaining bandwidth in the network in case of failures.

3.2.2 Adaptive routing

Adaptive routing is a technique where data is dynamically routed in a network based on real-time conditions such as traffic load, congestion, or failures [Rocher-González *et al.* 2022, Kasan *et al.* 2022, Rocher-Gonzalez *et al.* 2020, Zahavi *et al.* 2014, Geoffray & Hoefler 2008]. In adaptive routing, the routing tables of switches have multiple output ports for each destination, and the specific port used is dynamically selected according to the current network conditions. This contrasts with deterministic [Zahavi *et al.* 2009, Hoefler *et al.* 2008] or oblivious routing [Räcke 2009], where paths are predetermined and fixed, meaning each switch has a single output port that remains unchanged for each destination. Adaptive routing generally improves network performance and reliability by dynamically balancing traffic to reduce congestion.

Numerous deterministic routing solutions have been proposed for all-to-all exchanges [Zahavi 2011, Kumar & Kale 2004, Zahavi *et al.* 2009]. The debate between the use of adaptive versus deterministic routing has been discussed [Gomez *et al.* 2007, Rodriguez *et al.* 2009], and Infiniband, for example, offers an adaptive version of the deterministic routing algorithm Ftree [Nvidia 2023b]. However, adaptive routing did not seem suitable for our data acquisition network, as the bursty nature of the traffic prevents adaptive algorithms from adapting effectively, as shown in [Shin & Pinkston 2003]. Bursty traffic refers to peaks of data transmission that occur unpredictably and in short periods of time, followed by periods with low or no traffic. Adaptive routing makes routing decisions based on the state of the network. When a burst occurs, the adaptive routing algorithm may not have enough time to react to these sudden peaks, leading to inefficient routing decisions. The study in [Shin & Pinkston 2003] evaluates deterministic and adaptive routing with bursty traffic and shows that adaptive routing performance degrades more compared to deterministic routing. Furthermore, our performance evaluation in the studied data acquisition (DAQ) network, as presented in Section 4.2.2, proves that adaptive routing is not appropriate for our studied DAQ network.

3.2.3 Oblivious routing

Oblivious routing is a routing strategy where the routes for data transmission are set independently of the current network conditions, such as traffic load or failures [Räcke 2009]. In other words, the routing decisions are made without considering the state of the network at the time of transmission, making the routing oblivious to any changes in the network. This is in contrast to adaptive routing, where paths are adjusted dynamically based on real-time network conditions. The difference between oblivious and deterministic routing [Zahavi *et al.* 2009] is that oblivious routing employs multipath, meaning that the same source-destination pair can be routed with different paths, whereas deterministic routing always uses the same path for each source-destination pair.

Optimal oblivious routing often relies on linear programming techniques [Bienkowski *et al.* 2003, Prisacari *et al.* 2013c]. However, these solutions are challenging to apply in large-scale systems due to the significant computation time required for finding routing solutions, and they do not address the scheduling of the communications. In this thesis, we propose a scheduling algorithm designed to prevent congestion, which significantly reduces the complexity of the routing problem. This allows us to use a basic algorithm for simple failure scenarios and an Integer Linear Programming (ILP) model for challenging combination of failures. As a result, the combined scheduling and routing solution becomes feasible for the studied data acquisition network, with the computation of both routing and scheduling taking approximately 30 seconds in the worst case for challenging failure scenarios.

3.2.4 Weighted fat-tree routing algorithm

The Weighted Fat-Tree (WFatTree) routing algorithm, as proposed in [Zahid *et al.* 2015], is an adaptation of the Ftree routing algorithm [Zahavi *et al.* 2009], including advanced load-balancing techniques that improve throughput by up to 60% compared to Ftree in a fat-tree topology with 36 leaf switches [Zahid *et al.* 2016]. WFatTree achieves this by using weighted paths to more evenly distribute traffic in the network, preventing certain links from being overloaded while others remain underutilized. To be more precise, weights are assigned to the leaf switches based on the traffic each leaf switch will receive. However, this approach is not well-suited for handling all-to-all traffic patterns, where every leaf switch receives a roughly equal amount of traffic by the end of all phases.

Furthermore, as we demonstrate in Chapter 5, congestion is inevitable if the scheduling is not adapted to bandwidth reductions due to network failures. Even minor congestion can significantly degrade network throughput, as shown in Section 4.2.2. In this thesis, we propose a congestion-free routing and scheduling solution, unlike the WFatTree routing algorithm, which can still lead to congestion.

3.3 High performance computing topologies

In this section, we discuss various network topologies commonly used in data center networks that are well-suited for all-to-all traffic patterns. We discuss these topologies with a particular focus on fault tolerance.

3.3.1 Dragonfly topology

The all-to-all collective exchange has been extensively studied and applied in various widely-used topologies within HPC, such as the mesh [Sen *et al.* 2018] and torus [Yazaki *et al.* 2012, Doi & Negishi 2010] topologies. These topologies are still used and in place, but the most popular ones are currently Dragonfly [Kim *et al.* 2008] and Fat-tree for their scalability. These hierarchical topologies are characterized by the presence of multiple layers. Each layer is responsible for a certain scope of the network. Such topologies are easily scalable because multiple layers allow the network to be extended more easily.

Dragonfly topologies leverage the locality of data exchanges observed in some High-Performance Computing (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. Figure 3.1 illustrates a Dragonfly topology with three groups, each containing four switches. Each switch is directly connected to two servers. In each group, the switches are fully interconnected, allowing servers to exchange data simultaneously without congestion. However, each switch has only one direct link to a switch in each of the other groups. As a result, multiple hops are required for a server to communicate with another server in a different group. This can quickly lead

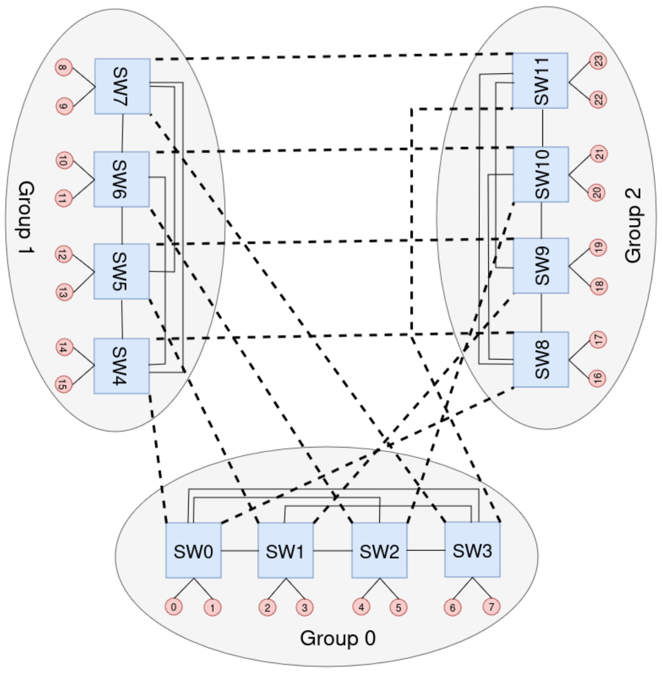


Figure 3.1: A Dragonfly topology with 24 servers and 12 switches. The topology is composed of three groups, with the number of switches in the same group denoted by α. Each group contains 4 switches ($\alpha = 4$), and the number of servers connected to each switch is denoted by p. Each switch is connected to 2 servers ($p = 2$). The switches in the same group are fully interconnected. Additionally, each switch has two links to external groups, represented by dashed lines, where the number of links to external groups is denoted by h, so $h = 2$. In a Dragonfly topology α, p and h can have any value. However, to balance the load in a Dragonfly topology, we need $a = 2p = 2h$ [Kim *et al.* 2008].

to congestion if many servers need to exchange data with servers in other groups, especially when there is no direct link between the source and destination switches.

Furthermore, the completion time of all-to-all exchanges with dragonfly topologies is increased compared to the application to fat-tree topologies [Prisacari *et al.* 2013a]. The Dragongly+ topology is a variation of the dragonfly topology in which nodes inside the groups are built as a two-layer fat-tree [Shpiner *et al.* 2017]. An analysis of performance variability of the dragonfly+ topology for all-to-all patterns has been conducted [Beni & Cosenza 2022]. The results revealed that the all-to-all collective exchange shows the most performance degradation compared to other communication patterns (one-to-all and all-to-one) when the exchanges are performed between different groups.

3.3.2 Demand-aware topology

A relevant approach to improve the bandwidth capacity in large-scale networks involves using demand-aware topologies [Griner *et al.* 2021, Zerwas *et al.* 2023, Avin *et al.* 2018, de O. Souza *et al.* 2022, Avin & Schmid 2021]. Demand-aware topologies refer to network topologies that are optimized based on the specific communication demands of the applications. Unlike traditional static topologies, which are fixed and do not change regardless of the traffic patterns, demand-aware topologies are more flexible and can adapt to the actual traffic matrix in the network. However, in the context of the studied data acquisition network, demand-aware topologies are not suitable, as the essential requirement of the data acquisition application is a continuous utilization of all-to-all exchanges and generally uses always the same traffic pattern without involving any other. Therefore, demand-aware topologies are not suitable for our problem, as traditional HPC topologies like fat-tree are already optimized for the all-to-all collective exchange.

3.3.3 HyperX topology

HyperX [Domke *et al.* 2019a] is a network topology designed for high-performance computing (HPC) and large-scale data centers. It extends the traditional mesh topology by allowing multiple links between switches, providing more flexibility and generally higher performance. HyperX is designed to be scalable, fault-tolerant, and suitable for various traffic matrix.

To our knowledge, the HyperX topology has never been evaluated with an all-to-all traffic matrix. However, two studies [Domke *et al.* 2019b, Lakhotia *et al.* 2021] have evaluated its performance with an all-reduce traffic matrix. Furthermore, one of these studies [Domke *et al.* 2019b] compares the HyperX topology to the fat-tree topology in terms of throughput and both studies show an improvement in achieved throughput.

Consequently, it could be interesting to evaluate the HyperX topology using the routing and scheduling solution presented in this thesis. The advantage of HyperX lies in its multiple redundant paths, which could simplify the optimization of our routing and scheduling solution in the event of failures.

3.3.4 The Fault-Tolerant Engineered Network topology

The Fault-Tolerant Engineered Network (F10) is composed of a variant of the fat-tree topology and a set of protocols that improve fault tolerance. In [Liu *et al.* 2013], the authors explain that the main weakness of the fat-tree topology lies in its symmetry; that is, each switch at layer l is connected in the same way to switches at layer $l+1$. This symmetry limits the diversity of available paths, reducing the fault tolerance of the fat-tree topology. To address this, the authors propose the AB fat-tree topology, where switches in a subtree are connected to switches in the upper layer using two different schemes: Scheme A and Scheme B. This approach breaks the symmetry of the fat-tree and increases the number of possible paths between two switches.

For our studied data acquisition network, the AB fat-tree topology is not useful because there are only two layers, meaning that each switch in layer 1 is connected to all switches in layer 2, resulting in the same number of paths between switches. However, for fat-tree networks with more than two layers, the AB fat-tree topology might offer advantages and could be worth testing with the routing and scheduling solution presented in this thesis.

3.4 Conclusion

In this chapter, we reviewed the literature related to our problem. We provided an overview of the various collective operations used in High-Performance Computing (HPC). We highlighted that the all-to-all operation is one of the most challenging because, although it uses the same bandwidth as the allGather operation, the number of different messages exchanged is higher, making the scheduling problem for all-to-all more challenging in terms of fault tolerance.

We also discussed how optimization of scheduling algorithms are used in Network-on-Chip (NoC). However, the traffic matrix in NoCs is not always all-to-all and focus on optimization of energy consumption or latency rather than fault tolerance.

We then present different routing strategies, such as scheduling-aware routing, which adapts routing based on communication schedules. Among the scheduling-aware solutions in the literature, one approach suggests increasing the number of phases to distribute congestion more evenly but does not achieve optimal number of phases to avoid congestion. In this thesis, we use this strategy and propose a lower bound on the number of phases in Section 5.4.1 to make it optimal and to avoid congestion.

Another routing solution discussed is adaptive routing, which creates multiple paths in the switch routing tables and adjusts them based on network conditions such as failures. We evaluate adaptive routing in Section 4.2.2 of this thesis and show that there are no improvements in the throughput achieved compared to deterministic routing.

Then, we discuss oblivious routing, which involves pre-computed paths with multiple paths between a source and a destination. Oblivious routing algorithms often rely on linear programming to optimize routing in terms of bandwidth or fault tolerance. Linear programming often involves significant computation time, making the oblivious routing solutions often less applicable to large networks. However, we use this strategy and present an evaluation of our Integer Linear-Programming (ILP) model to compute the routing in Section 6.6.3 of this thesis. The computation time we achieve makes our routing solution applicable for our data acquisition network.

We also present the weighted Fat-tree (wFatTree) routing algorithm, an adaptation of the Ftree routing algorithm proposed by InfiniBand, which improves the load-balancing on the links. However, wFatTree does not seem effective for the studied network because it prioritizes certain communication flows based on the traffic each switch connected to the servers receive. In the LHCb DQA network, all switches connected to the servers receive roughly the same amount of traffic, which makes this approach less suitable.

Finally, we discuss various network topologies relevant to our problem, including the Dragonfly topology, which is not optimal for all-to-all exchanges due to its blocking nature except in intra-groups. Demand-aware topologies adapt to the traffic matrix used and are suitable for networks with diverse traffic patterns but not for data acquisition networks that typically use the same traffic matrix all the time. We introduce the HyperX and AB fat-tree topologies, which offer improved fault tolerance by increasing the number and diversity of paths available between switches. These topologies are interesting solutions for our problem and could be compatible with our proposed scheduling and routing solutions.

Chapter 4

Study of failures in the LHCb DAQ network

Contents

In this chapter, we delve into the motivations driving our research problem by showing how often failures occur, how long they last, and their impact on throughput with various routing and scheduling solutions.

Past studies have shown that network failures in data centers are a common occurrence [Singh *et al.* 2021], as evidenced by the study conducted by the authors in [Gill *et al.* 2011]. The authors propose an analysis of network failures in data centers, presenting statistical insights derived from a year-long observation of a real data center network. Despite the generally robust infrastructure of data center networks, the findings underscore the frequency of failures on a daily basis throughout the measurement period. Furthermore, the study shows the long duration of certain link failures, often attributed to repair difficulties such as limitations in spare network equipment or limited accessibility to the affected link.

Furthermore, a time correlation exists among failures in large-scale distributed systems. Yigitbasi et al. [Yigitbasi *et al.* 2010] show the presence of peak periods, characterized by high failure rates that can persist for several hours. Their analysis shows that, on average, these peak periods account for over 50% and in some cases up to 95% of the system downtime.

In this chapter, we present statistics on the duration, frequency, and underlying causes of network link failures. We show that failures are frequent and can last a long time. Throughout this thesis, we were able to monitor and recover data on network failures that occurred over a two-month period of activity when the Large Hadron Collider (LHC) was operational and data were exchanged in the DAQ network. During this period, we closely monitored the causes of these failures. Notably, test days were conducted, leading to network failures.

Subsequently, we proceed to evaluate the throughput achieved during these failures. We observe that link failures can significantly degrade performance, as bandwidth utilization in the DAQ network is close to maximum capacity. A single failure leads to congestion, reducing the throughput from approximately 46 Tbps to 30 Tbps in total with the best routing and synchronization approach.

First, we propose to evaluate the effect of various InfiniBand routing algorithms, suitable for a fat-tree topology, on the throughput. We show that, depending on the algorithm used, some create more congestion than others, both when failures occur and when the network is operating normally. Based on the results, we can assess which routing algorithms perform best under normal conditions and during failures.

Then, we propose to evaluate the throughput achieved with and without synchronization of the all-to-all collective exchange. The non-synchronized MPI application was developed by us to perform our measurements. While previous investigations [Pisani *et al.* 2023a] have shown performance differences between synchronized and non-synchronized all-to-all exchanges on our network, our contribution introduces novel measurements, particularly in the context of performance in case of failures, which have never been measured on our network. To conclude this chapter, we derive some design recommendations.

4.1 Statistics of failures

The literature shows that failures in large-scale networks are frequent and can persist for an extended duration due to the difficulty to repair network failures [Singh *et al.* 2021]. In this chapter, our contribution is to gather and analyze failure statistics in the Event Builder network of the LHCb experiment to understand the duration, frequency, and nature of network failures in this network. Our statistics cover a two-month period during which the Large Hadron Collider was operational, and physics data was transmitted within the Event Builder network.

4.1.1 Methodology

The failures are monitored using OpenSearch, an open-source tool derived from Elasticsearch and maintained by a community of contributors [OpenSearch 2024].

OpenSearch comprises various components, including dashboards for data visualization and analysis, along with other tools for managing and querying data. To monitor the failures, Logstash, an open-source data processing pipeline [Elasticsearch 2024], collects and processes data from the system log messages on a server running OpenSM. The collected data are fed into OpenSearch and include all possible logs from OpenSM, such as changes in topology, routing engine updates, and the status of the subnet. Specifically, we are interested in logs that indicate changes in switch port state, which are provided by messages of type "osm_spst_rcv_process" that contains the switch ID, port ID, and port state.

For instance, when a link goes down, the switch port connecting that link goes from the ACTIVE to DOWN state. Conversely, when the link is repaired, it goes from DOWN to INIT (and then ACTIVE). To obtain this data, we create a query in OpenSearch targeting these specific logs, executing it automatically every day for two months. The output is a Comma-separated values (CSV) file containing all the necessary information for failure analysis: timestamp, involved switch and port, and the change in port state. Subsequently, we parse these files using a Python program to analyze the data.

4.1.2 Duration

To properly evaluate the duration of failures, we considered the presence of flapping links. Flapping links are common in networks in general, and can cause considerable perturbations to networks due to multiple consecutive route re-configurations [Merindol *et al.* 2018].

A flapping link can be defined as a link that oscillates repetitively between a down and up state. In this thesis, we are particularly interested in the interval between flaps, with a flap being defined as a link that goes from up to down and back within a few seconds. The multiple re-configurations caused by these flaps create instability and can degrade significantly the throughput, as the state of the link can oscillate continuously over long periods, sometimes for several hours. Consequently, this series of short-duration failures (typically lasting only a few seconds) can, in fact, represent a prolonged failure that can last for several hours.

To consider this phenomenon in our analysis of the duration of failures, we classified the observed flapping links and their flapping periods over the two-month measurement period. We define a flapping period as a period during which multiple flaps are observed on the same link over a relatively long duration, which can last from several minutes to several hours. We experimentally determine the most probable duration between flaps. To be more specific, Figure 4.2 shows the distinction between the duration of a flap and the duration between two flaps. In the cumulative distribution presented in Figure 4.1, we only consider the interval between flaps, which is $T3 - T2$, represented by a dashed arrow. In Figure 4.3, we consider the duration of the flaps (or failures in general) which is the duration $T2 - T1$, represented by a solid arrow.

The results are illustrated in Figure 4.1, we show the cumulative distribution

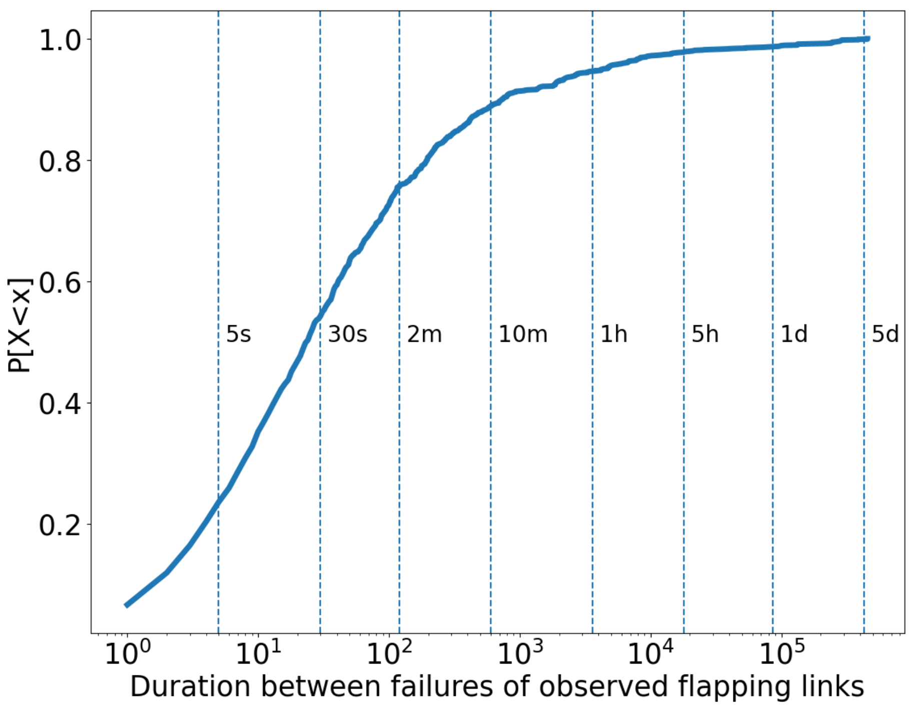
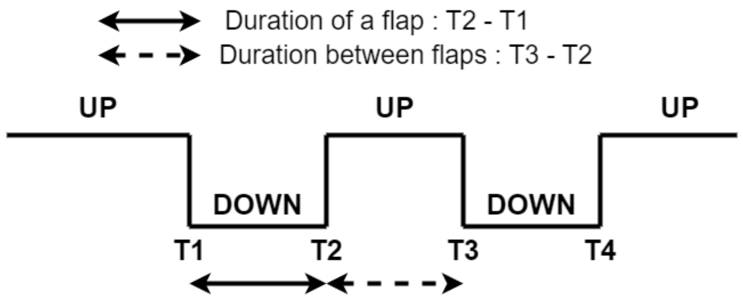


Figure 4.1: Cumulative distribution of the interval T3 - T2 (in Figure 4.2) between flaps of observed flapping links and their flapping periods.



Duration of a flap : T2 - T1
Duration between flaps : T3 - T2
UP
UP
UP
DOWN
DOWN
T1
T2
T3
T4

Figure 4.2: Distinction between the duration of a flap and the interval between two flaps. The duration of a flap is represented by the solid arrowed line and the value is T2 - T1. The interval between two flaps is represented by the dashed arrowed line, which is T3 - T2.

of the duration between failures on a same link. In the Event Builder network, we observe that when a link starts flapping, 90% of the failures observed during the flapping period of the observed flapping links have a time interval of less than 10 minutes. This means that when the first failure occurs on the link during its flapping period, the next one probably occurs less than 10 minutes later.

Based on this insight, we consider a flapping link period as a single failure by ignoring the time interval between two failures that are inferior to 10 minutes. This means that when two or more failures occur on a flapping link and their time interval is less than 10 minutes, these failures are considered as one, and the failure period starts at the time the first failure occurred until the last one is recovered. To be more specific, in Figure 4.2, if the interval $T3-T2$ is less than 10 minutes, the failure with a duration of $T2 - T1$ and the subsequent failure with a duration of $T3 - T2$ are considered a single failure, with the total duration being $T4 - T1$. By applying the flapping links correction, we discovered that nearly 70% of the failures result from flapping links. Over the 2605 observed failures, 1778 are attributed to flaps. After applying the correction, these flaps were reduced to 287 failures, representing 287 periods during which the links were flapping.

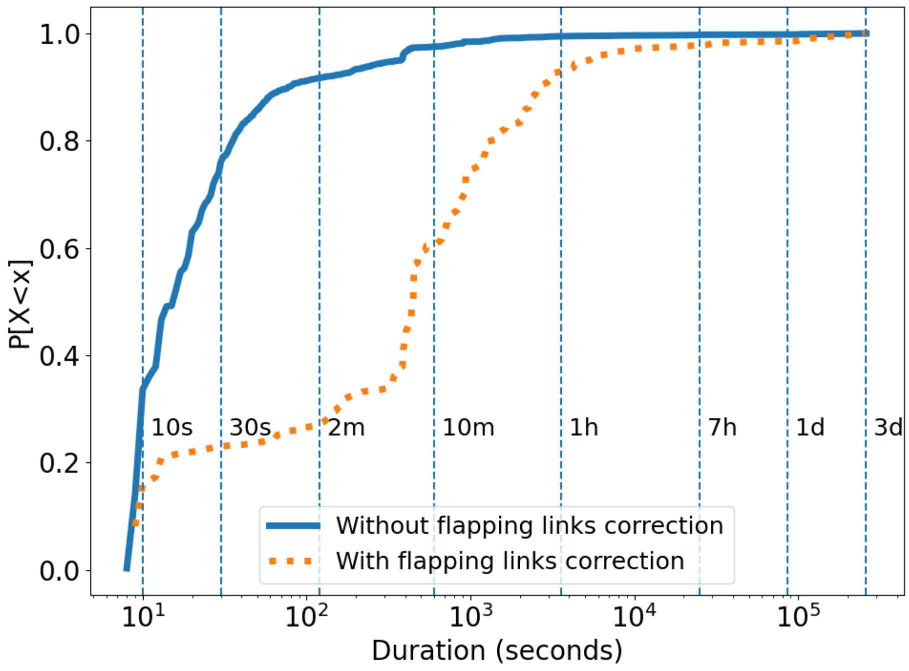


Figure 4.3: Cumulative distribution of failure duration. The blue line represents the cumulative distribution of failure duration without the flapping links correction. The orange dotted line represents the cumulative distribution of failure duration with the flapping links correction. Flapping links correction is applied by counting failures of a flapping link as one long failure rather than several small failures spaced over a short period.

Figure 4.3 illustrates the cumulative distribution of link failures duration observed throughout March and April 2024 on the Event Builder network. The duration of a link failure refers to the period during which a link transitions from an operational state to a non-operational state until it returns to the operational state. In Figure 4.2, it corresponds to the duration $T2 - T1$. The blue line shows the results for all observed failures, excluding periods during which performance tests were conducted on the Event Builder network. These tests were generated by ourselves and involved manually disabling links to evaluate how the network reacts to failures. We chose to remove them as they bias our experiments. Given that the duration of these failures is predetermined and programmed, they are irrelevant to the study. Otherwise, all the links between the leaf and spine switches, as well as the entire period between March and April 2024, are considered in the cumulative distribution of link failures duration. The dotted orange line represents the cumulative distribution of link failure duration when applying the flapping links correction.

In the results without the flapping links correction (the blue line), we observe that instances of very short failures are not frequent, 33% of the observed failures have a duration of less than 10 seconds. However, the data reveals that 88% of observed failures are short-lived, lasting less than a minute, while 97% conclude within 10 minutes. Consequently, the majority of failures are brief, with the longest failures lasting around 3 days.

Compared to the duration of failures without the flapping links correction, failures duration are prolonged when the correction is applied (the orange dotted line), with 24% of the failures lasting less than 1 minute and 60% lasting less than 10 minutes. This highlights the severity of flapping links as a network issue, as they often lead to prolonged failures. Furthermore, the impact on the network is even worse because each flap requires computing new routes and updating routing tables. These actions consume time [Dandapanthula 2011, Francois & Bonaventure 2008], approximately 1.5 seconds in our network. Consequently, these failures can degrade network performance and cause long-lived performance degradation. A more effective strategy might be to disable a flapping link as soon as it starts to flap, as this is likely to result in better overall throughput by allowing the network to adapt to the failure, rather than repeatedly restarting the link.

4.1.3 Frequency

Figure 4.4 illustrates the distribution of failures over the two-month period of March and April 2024. Only the days and links where failures occurred are represented in Figure 4.4. We observed 28, not necessarily consecutive, days without any failures over the two-month period. Among these, there is a maximum of 6 consecutive days without failures. Each link was assigned an ID based on the order of the switch ID and the port ID, which were sorted in ascending order. In total, 150 links in the Event Builder network failed at least once during the month of March and April 2024. The test days for the Event Builder network occurred on 03/08, 03/10, 03/11 and 03/12, which explains the occurrence of failures for various links on these days. During these tests, various links were manually disabled, which explain why a lot of links experienced a few failures on these days. These are the events removed from Figure 4.3. Furthermore, two links experienced repeated failures throughout the

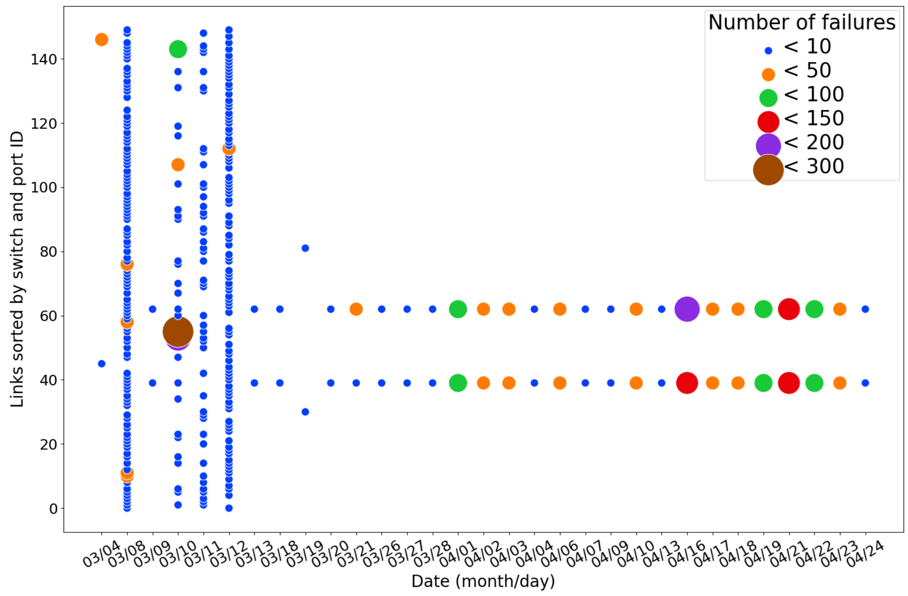


Figure 4.4: Distribution of network link failures over a two-month period. The x-axis represents the days in March and April 2024 when failures occurred. The y-axis represents the IDs of the links that failed during this period. The link IDs are assigned based on the switch IDs and the port IDs which are sorted in ascending order.

entire period. These links represent a single flapping link in two directions. The link frequently experienced flapping, with failures occurring nearly every day and recording more than 200 failures on a single day (April 16 - 04/16). As detailed in Section 4.2.2, the consequences of such failures can significantly impact network performance, resulting in an important loss of throughput. Unfortunately, this link could not be repaired during the measurement period due to the unavailability of spare equipment and the location of the link.

4.1.4 Nature of failures

During March and April 2024, we observed a total of 2605 failures, out of which 549 were attributed to tests conducted on the Event Builder network, while 2056 were real network failures. Among the 2056 real failures, 1778 are due to flapping links. Flapping links are caused by cable deterioration over time. Excluding the flaps, there remain 278 isolated failures, which are generally due to hardware issues such as dirty optical fibers or software problems. Additionally, maintenance activities were conducted on the Event Builder network servers, resulting in a total of 332 failures of servers. However, failures of servers are not relevant to our study, as they involve complete disconnections of servers from the network. There is no means of recovering from a server failure with the current infrastructure. In such cases, the available bandwidth remains sufficient for the all-to-all exchange as the number of sources is reduced.

4.2 Throughput achieved by Infiniband routing algorithms

In this section, we present our measurements of the throughput obtained on the DAQ network of the LHCb experiment using the various Infiniband routing algorithms relevant to a fat-tree topology.

4.2.1 Methodology

The links that were deactivated were chosen randomly and included 1 to 10 simultaneous failures. We chose these numbers of failures to show the gradual degradation of throughput in the event of failures for the different routing algorithms. Since the all-to-all application is synchronized, all servers experience the same degradation in bandwidth, even if the switch to which they are connected does not have any failed links.

The routing algorithms we evaluate are Ftree, Ftree adaptive routing, Ftree to Min-Hop, Min-Hop, Up-Down and Up-Down adaptive routing. Ftree is the routing algorithm used for fat-tree topologies and shifted communication patterns such as the linear-shift pattern [Zahavi *et al.* 2009]. The Ftree routing algorithm, coupled with the linear-shift, ensures that there will be no congestion on the network if the bandwidth available is sufficient as explained in Section 2.3.3. In the event of failures, the default behavior of OpenSM, the subnet manager, is to switch to the Min-Hop routing algorithm, as the Ftree algorithm is not suited for non-pure fat-tree topologies. The Min-Hop algorithm computes the shortest path between a source and a destination while distributing the load as evenly as possible. Up-Down is similar to Min-Hop but prevents deadlocks in the network, it offers less choice in the paths to use as they are constrained by ranking rules to avoid deadlocks. Adaptive routing for Up-Down and Ftree are algorithms that act like Up-Down and Ftree respectively, except that for each communication, several routes are computed to balance the traffic if a link is congested [Nvidia 2023b]. Other Infiniband routing algorithms exist, such as LAyered SHortest Path Routing (LASH) and Dimension Order Routing (DOR), but they are not suitable with our traffic or topology. LASH requires virtual lanes to create multiple channels within a physical link, which is not suitable for our topology as each link needs to be used at full capacity for each communication flow. DOR is used for k-ary n-cubes topologies [Nvidia 2023b].

To configure each routing algorithm, we manually modified the *routing_ engine* parameter in the OpenSM configuration file to specify which algorithm to use. To use the Ftree algorithm all the time, even in the event of a failure, and thus avoid switching to Min-Hop, which is the default behavior, we also added a *root_ guid_file* containing the list of GUIDs for the spine switches. This file is also used for the Up-Down routing algorithm as it needs the root nodes to create the ranks. This file allows the subnet manager to trust us on the number of active spines with no failures, so that we can always use Ftree as the routing algorithm, even though it's only designed for pure fat-tree topologies. For adaptive routing, we changed the *AR_ENABLE* parameter from *false* to *true* to enable adaptive routing on the switches. The configuration of OpenSM is done before the measurement. Following every configuration change, we also manually restart OpenSM and check that all the changes are taken into account, which takes a seconds.

To illustrate the performance degradation during failures, the status of the ports on the switches connecting the links involved in the failure scenarios was set to DOWN before the measurement. To measure the throughput, we use an existing Python program that counts the number of events reconstructed by each server over 5 seconds in the all-to-all application and records the data in a Comma-Separated Values (CSV) file. The duration of a measurement was 60 seconds, which creates 12 measurement points in the CSV file. Then, we used a Python program to parse the output CSV file and obtain the number of events reconstructed for each measurement point and each server. We then multiply this number by the size of an event and convert the result into bits. This results in the throughput of the server in bps, which we then convert to Gbps.

4.2.2 Results

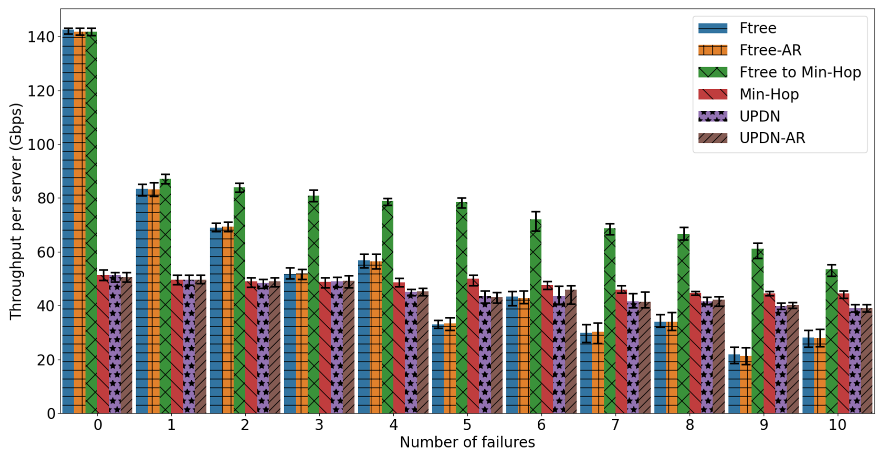


Figure 4.5: The throughput achieved per server according to the failure scenario with the Infiniband routing algorithms suitable for a fat-tree topology. The routing algorithm in the evaluation are Ftree, Ftree adaptive routing, Ftree to Min-Hop (default behavior), Min-Hop, UPDN, UPDN-AR. To obtain the 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.

Figure 4.5 shows the throughput achieved by each server. Without failures, the best routing algorithm is Ftree and its variants. This is because Ftree, coupled with the linear-shift pattern, creates no congestion on the network, as no link is used twice in the same direction. In contrast, Min-Hop and Up-Down create congestion because they are not adapted to the communication pattern; several links are used multiple times in the same direction, creating congestion which significantly impacts throughput.

Ftree to Min-Hop, the default behavior of the InfiniBand controller, shows the best performance compared to all other routing algorithms. When there are no failures in the network, Ftree is the most optimized routing algorithm for our traffic matrix and fat-tree topology. In case of failures, the Min-Hop routing algorithm takes over and distributes the load between links that are affected by congestion due to failures. The combination of these two algorithms thus provides the most optimized routing solution for our DAQ network. However, it should be noted that we observe a significant degradation in throughput, dropping from approximately 144 Gbps to 87 Gbps with just a single network failure, and continuing to degrade slowly as the number of failures increases. This is because the network operates close to 100% bandwidth utilization under normal conditions. Therefore, even a single failure causes significant congestion in the all-to-all traffic matrix, impacting all servers.

Ftree has variable throughput as the number of failures on the network increases. In the event of failures, congestion is inevitable, since bandwidth utilization is close to 100% when there are no failures on the network. Therefore, even a single failure creates unavoidable congestion. Ftree is not suitable for failures, as the load will be less spread over the different links, creating more congestion than necessary. In particular, after three failures, throughput varies significantly, and the scenario with four failures has a better throughput than the one with three failures. This result is attributed to the fact that Ftree no longer effectively balances the traffic after a certain number of failures. With Ftree, when a link l_0 is down, the link l_1 connected to the next port takes the load of l_0 which creates even more congestion on the link l_1. This creates a significant decrease in throughput. To summarize, Ftree, in the absence of failures, evenly distributes the traffic and does not create conflicts when it is combined with the linear-shift scheduling, resulting in the best achievable throughput. However, in case of failures, Ftree changes its routes from the broken port to the next available port, which creates significant congestion.

Ftree/Ftree-AR and Up-Down/Up-Down-AR offer similar performance with and without failures. Adaptive routing is expected to be more efficient during network failures, as it dynamically redirects traffic to avoid congestion. This is attributed to the fact that the traffic generated by the DAQ application is too bursty, compared to datacenter traffic in general [Abdous *et al.* 2021, Zhang *et al.* 2017, Roy *et al.* 2015], for the Ftree adaptive routing algorithm to adapt accordingly. Bursty traffic is characterized by unpredictable peaks in data transmission rates that can last for a short period of time, in our case, in the order of milliseconds. Beyond our measurements, another source has indicated that adaptive routing leads to more performance degradation for bursty traffic compared to deterministic routing [Shin & Pinkston 2003].

Without failures, Min-Hop performs poorly compared to Ftree to

Min-Hop. Without any failures, Min-Hop performs really poorly with an average throughput of 53 Gbps per server. This is explained by Min-Hop being not suitable for the linear-shift pattern which creates a lot of congestion on the links. In the event of failures, Min-Hop alone does not perform as well as when the controller switches from Ftree to Min-Hop which is surprising as this is the same algorithm that is used (Min-Hop in case of failures). These different results can be explained by the fact that the controller (OpenSM) always preserves the existing routing in cases where there are no changes in the connected switches within the topology [Nvidia 2023b]. This means that if a link is added or removed, OpenSM will not recompute routes that do not need to change. Consequently, Min-Hop routing without failures and Min-Hop routing with a failure do not change significantly, because only the traffic coming from or going to the leaf switch with the failure needs to be balanced, and a significant part of the routing is based on Min-Hop without failures. As we have observed, Min-Hop without failures presents a lot of congestion because it is not adapted to the linear-shift pattern. While Ftree to Min-Hop routing is based on Ftree, which is optimized for the linear-shift pattern, and only the routes of communication flows going to or coming from the failed switches are balanced. Therefore, Ftree to Min-Hop shows much better performance in the event of a failure than Min-Hop, even though the same routing algorithm is used.

Up-Down presents similar or lower performance than Min-Hop. This is due to Up-Down algorithm having fewer route choices available. Its principle is to avoid certain routes to prevent deadlocks from occurring in loops within the subnet. In a network, a loop-deadlock occurs when data packets can not be sent because each packet is waiting for another to release a resource or complete an operation, creating a circular dependency with no resolution. The Up-Down routing algorithm uses a hierarchical structure on the network with a spanning tree and a node ranking system. Paths are determined by following an upward and a downward phase through the tree. [Nvidia 2023b] These properties reduce path diversity and can constrain them, resulting in more congestion compared to Min-Hop, especially when the number of failures is higher.

To conclude, the Ftree to Min-hop approach is clearly the most effective compared to other routing strategies. However, there is still room for improvement, as a single failure significantly impacts throughput, reducing it from approximately 144 Gbps per server to 87 Gbps with the best strategy. In Chapter 6, we propose a new routing approach that more gracefully handles failures.

4.3 The effect of synchronization on the throughput

In this section, we investigate the effect of synchronization on the throughput in nominal state and upon failures. Our objective is to devise efficient communication strategies in the two situations (with and without failures). The all-to-all synchronized application currently in production and used by the Data Acquisition (DAQ) system has been developed internally for the LHCb experiment[Pisani *et al.* 2023a].

We evaluate the performance of synchronized all-to-all compared to the non-synchronized as we ask ourselves whether synchronization always remains the optimal solution. For this purpose, we develop a new all-to-all application without synchronization that supports the same throughput as the synchronized one currently in production.

4.3.1 The non-synchronized All-to-All MPI application

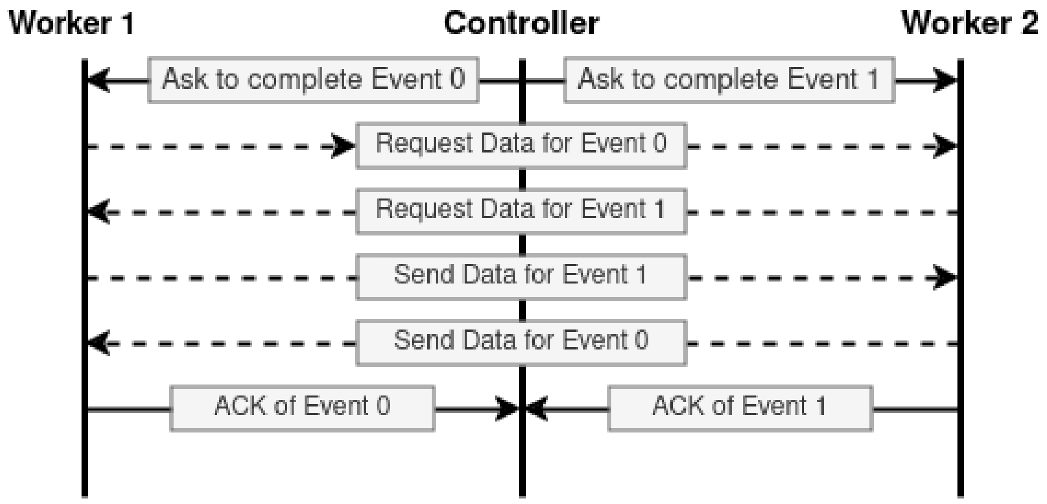


Figure 4.6: Description of the MPI all-to-all application without synchronization.

Both the non-synchronized and synchronized all-to-all applications are developed using the Message Passing Interface (MPI). MPI is a tool for high-performance scientific computing that provides syntax, semantics, and libraries to enable multiple processes to communicate with each other. MPI relies on Single Program Multiple Data (SPMD) principle, which allows the user to write a single program to be executed by a set of processes, each with a distinct role [MPI 2021]. The user's challenge lies in defining how these processes communicate with each other based on their roles within the set. Additionally, MPI provides a set of functions to facilitate communication between processes, such as MPI_Send and MPI_Receive. These functions enable processes to send or receive data. While MPI_Send and MPI_Receive are blocking functions, requiring all data to be sent or received before proceeding to the next instruction, MPI_IReceive and MPI_ISend are non-blocking, allowing other instructions to be executed even if all data has not yet been received or sent. In our non-synchronized application, illustrated in Figure 4.6, we use only the non-blocking functions MPI_IReceive and MPI_ISend for performance reasons. Blocking functions can slow and block the traffic in the all-to-all exchange, which is undesirable in a non-synchronized application. Furthermore, MPI offers a variety of functions for performing collective exchanges between processes, such as broadcast with MPI_Bcast or gather with MPI_Gather. MPI also provides an all-to-all exchange function with MPI_AlltoAll. However, this exchange is synchronized.

Synchronization involves dividing the all-to-all exchange into phases. At each phase, every process communicates with a single process until each process has communicated with all others. The different processes synchronize with each other, meaning that once one process has finished sending its data, it must wait until all other processes have finished sending their data before proceeding to the next phase. This synchronization is achieved using the MPI_Barrier function. In this function, once a process completes its exchange, it sends a notification and then proceeds to wait. Depending on the chosen Barrier algorithm, the notification can be centrally transmitted or distributed among different processes [Hensgen *et al.* 1988].

In the centralized approach, a single process acts as a controller and receives the notifications from the other processes once their exchanges are finished. Once all processes have completed their exchanges, the controller notifies all processes to proceed to the next phase [Hensgen *et al.* 1988].

In the distributed approach, known as Tournament, a binary tree structure is implemented among the processes [Hensgen *et al.* 1988]. At each layer and for every pair of nodes in the tree, a winner is designated and takes responsibility for notifying the winner at the higher level of the tree. Once all notifications have been sent to the global winner, it then propagates the notification to allow the processes to proceed to the next phase, and subsequently, the notification is propagated to the layers below in the same way.

The LHCb DAQ network makes use of the Tournament algorithm. This barrier algorithm was selected over the centralized algorithm due to its better performance[Pisani *et al.* 2023a]. Since we aim to evaluate the performance of our network with a non-synchronized collective exchange, we need to develop our own all-to-all exchange without synchronization. As the synchronized all-to-all application used by the LHCb DAQ system relies on MPI, we chose to develop the non-synchronized application using MPI to ensure the two applications are comparable and to take advantage of the simplicity of using MPI.

In the non-synchronized all-to-all application, we simulate the Event Building process similarly to the synchronized one, which involves reconstructing collision events generated by the Large Hadron Collider. Each server in the Event Builder network has one process of the MPI application allocated. The principle of the non-synchronized all-to-all application is that one process serves as the controller while all others act as workers. The controller assigns each worker a collision event to reconstruct. Each worker has a piece of data of every event and sends it to the worker responsible for reconstructing that event. Figure 4.6 illustrates our non-synchronized application with one controller and two workers. The controller assigns worker 0 to reconstruct the event with ID 0 and worker 1 to reconstruct the event with ID 1. Subsequently, worker 0 and worker 1 exchange requests for data on the corresponding event. Upon sending the data and reconstructing the event, the workers send an acknowledgment to the controller, prompting the assignment of subsequent events to be reconstructed.

In this approach, there is no scheduling; when workers are assigned an event by the controller, they send a request to receive data for this event to all workers in the order of their ID. However, the data request is non-blocking, which means that all sources may send their data concurrently. The only blocking condition for a worker is that it is required to receive all data from other workers on the event it needs to reconstruct before making a request to the controller to assign another event.

4.3.2 Experiment setup

To compare the synchronized and non-synchronized all-to-all versions in the event of failures, we manually deactivated the links between the leaf and spine switches. We did not test disabling the links between leaf switches and servers, as these do not affect the bandwidth of the all-to-all exchange for the remaining nodes.

The links that were deactivated were chosen randomly and included 1, 3, and 5 simultaneous failures. We chose this number of failures because the literature has shown that groups of failures containing more than 5 failures are unlikely, with only 10% of groups containing more than 4 failures [Singh *et al.* 2021]. Our network shows the same behavior, with a median of only 1 simultaneous failure. The maximum number of coexisting failures observed is 5.

For each failure scenario, the tests were repeated randomly 10 times. To measure the throughput, we use the same method as described in Section 4.2.1.

The routing algorithms used during these tests are Ftree and Min-Hop. Ftree is used when there is no network failure. The network switches to Min-Hop when a failure occurs. We chose these algorithms because, firstly, they are the default behavior of OpenSM and, secondly, they are recommended because they offer the best performance for our topology, as shown in Section 4.2.2.

4.3.3 Scalability of synchronized and non-synchronized all-to-all

We first propose to re-evaluate the scalability of the synchronized and non-synchronized all-to-all exchange of [Pisani *et al.* 2023a] due to significant changes in the DAQ application since this publication, potentially leading to new results.

We evaluate the scalability of the current network design by measuring the global event building throughput when enabling an increasing number of servers in Figure 4.7. To increase the number of servers, racks of servers are consecutively added to the system. Each rack contains one leaf switch and 16 to 20 servers.

In our evaluation, synchronization proves to be more advantageous. As the system grows in size, synchronization enables to reach higher throughput. When the DAQ system is used at full capacity, we observe a throughput gain of about 15 Tbps over the non-synchronized all-to-all. By contrast, in the prior test [Pisani *et al.* 2023a], the throughput gain was approximately 5 Tbps at full capacity of the Event Builder network with 326 servers.

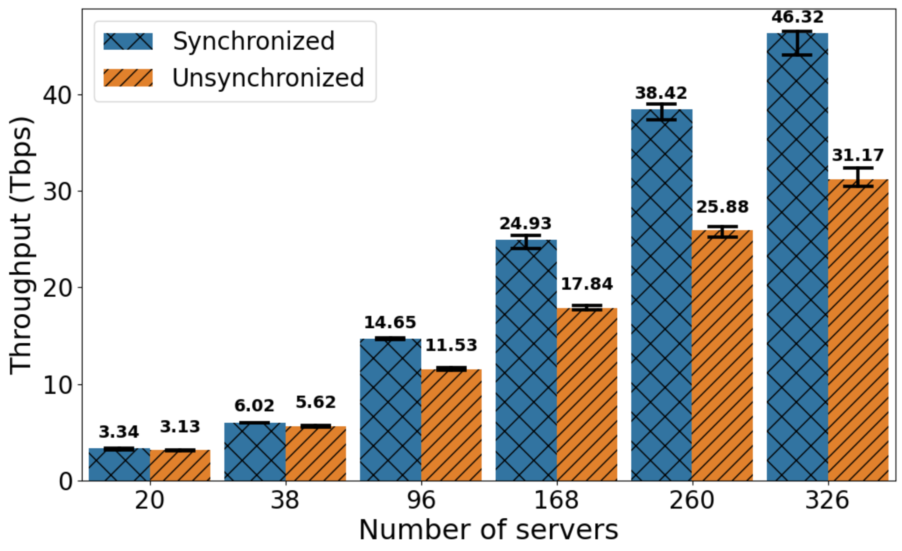


Figure 4.7: Scalability of the synchronized and non-synchronized all-to-all applications. The error bars represent the minimum, mean and maximum values.

The difference in performance between synchronized and non-synchronized all-to-all arises from the assurance provided by synchronization that the scheduling of communications during each phase is always respected. With synchronization, every server waits until all others have transmitted and received their data before proceeding to the next communication. This approach effectively prevents network congestion, as proper scheduling and sufficient bandwidth ensures that there will be no congestion on the network links, thus improving throughput. Conversely, in the non-synchronized all-to-all, there is no scheduling as the exchange is not synchronized, which creates network congestion and reduces the achieved throughput. Therefore, the synchronized all-to-all provides better performance, particularly in systems used at full capacity. Although there is cost in synchronization time as the sources need to wait on each other, this cost can be offset by the performance gained by avoiding congestion on the network.

4.3.4 Throughput achieved by the synchronized and non-synchronized all-to-all in case of failures

The synchronized approach performs better than the non-synchronized approach without failures and when the system is used at full capacity. In this section, we evaluate the performance of these two approaches in the event of failures, considering that link failure events are common, as demonstrated in Section 4.1. This evaluation has never been conducted before.

The throughput of the synchronized exchange can significantly decrease in the event of failures. Figure 4.8 shows the results of the evaluation of

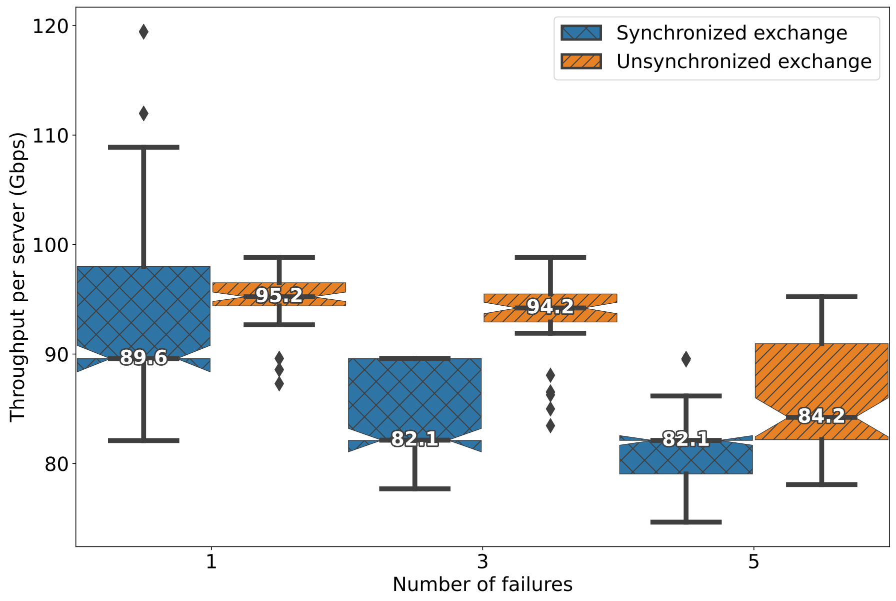


Figure 4.8: Synchronized and non-synchronized all-to-all exchange throughput per server as a function of the number of failures. Failure scenarios are randomly generated 10 times for each number of failures. There are 326 servers in the topology. The number of simultaneous failures in the Event Builder network is 1, 3 or 5. The boxplots represent the minimum, 25th percentile, median, 75th percentile, and maximum values. Outliers are also depicted.

the synchronized and non-synchronized approaches in the event of failures. The failure scenarios have been replicated 10 times for each number of simultaneous failures. The links taken down are picked randomly. In total, there are 30 experiments. In particular, we demonstrate that throughput can significantly decrease in the event of failures, from a median of 142.71 Gbps without failures to 89.6 Gbps for the synchronized exchange, even with just one failure in the entire network of 360 optical links. The variation in results for the same number of failures for synchronized and non-synchronized exchanges are due to some links carrying more flows than others. When they are down, more traffic is impacted by their failure. For instance, the minimum achieved throughput per server for a single failure with synchronized exchange is 82 Gbps, while the maximum value is 120 Gbps. For the 82 Gbps result, the disabled port of one switch served as the output port for 26 destinations in all phases, whereas for the 120 Gbps result, the disabled port on another switch was used as the output port for 9 destinations. Consequently, disabling the port used to reach 26 destinations, which support more communication flows, had a much greater impact on the throughput compared to the port used for 9 destinations, resulting in a decreased throughput in case of failures.

Synchronized exchange shows lower performance compared to non-synchronized exchange in the event of failures. In the synchronized all-to-all, the cost of synchronization can be offset by the performance gained from avoiding congestion on the network. However, this approach lacks adaptability in the event of network link failures because its performance is dependent on avoiding congestion. When congestion becomes unavoidable due to link failures with the currently used routing and scheduling, performance suffers accordingly. Contrarily, the non-synchronized approach shows better adaptability in case of failures. The lack of synchronization points allows it to make efficient use of the remaining network capacity as we highlight in the next section.

4.4 Design recommendation

In our measurements, we demonstrated that the non-synchronized version of the all-to-all exchange performs better than the synchronized version in the event of failures. As discussed in Section 4.1, some failures can be long-lived failures due to flapping links. For instance, during two months of measurements, a single optical link failed 861 times, resulting in a total downtime of approximately 6.1×10^5 seconds, which represents approximately seven days and one hour. In Section 4.3.4, we show that the median throughput of the synchronized exchange for a single failure is 89.6 Gbps per server, whereas the non-synchronized application achieves a median throughput of 95.2 Gbps per server. Consequently, the total application throughput during a failure is 29.2 Tbps for the synchronized application compared to 31 Tbps for the non-synchronized application. We obtain these numbers by multiplying the median throughput for a server by the total number of servers, which is 326. This means that every time one link fails, the synchronized application loses 1.8 Tbps compared to the non-synchronized one, which represents a total loss of approximately 1.1×10^6 Tb over the months of March and April, given the flapping link's total downtime of 6.1×10^5 seconds.

4.5 Conclusion

In this chapter, we present statistics about failures collected over two months. Our results revealed that failures occur frequently, with 2605 failure events recorded during the data taking period. Most of these failures were due to flapping links, which are characterized by fluctuations between operational and non operational states, leading to extended periods of downtime. This highlights the existence of long-lived failures in the studied network, which cause a significant loss of throughput.

Then, we evaluate the throughput achieved by relevant Infiniband routing algorithms for our studied network. We elaborate on the results obtained by highlighting the specific properties of each routing algorithm.

Finally, we evaluate two alternative approaches to Event Building on the LHCb DAQ system, showing that a synchronized approach can reach high throughput in normal conditions. However, upon link failures, the synchronized approach faces significant performance reduction. In these scenarios, eliminating synchronization is a quick solution to reduce the performance degradation. Furthermore, as we demonstrated the existence and impact of flapping links, we believe that a more effective strategy would be to disable a flapping link as soon as it starts to flap. Instead of repeatedly restarting the link, which can trigger multiple network reconfigurations and cause instability, it would be better to use the non-synchronized all-to-all exchange to adapt to failures.

Chapter 5

Fault-Adaptive Scheduling Algorithm

Contents

In this chapter, we present our scheduling algorithm, designed to adapt to failures in a fat-tree topology, ensuring that network congestion is avoided. We start by introducing the Latin Squares and how they can define an all-to-all communication pattern. We also introduce the concept of bandwidth reduction which is one of our contribution. Then, we study a scheduling algorithm that demonstrates potential in addressing failure scenarios by limiting the use of bisection bandwidth. This algorithm is derived from the formulas described in the referenced paper [Prisacari *et al.* 2013b] and, like the linear-shift scheduling [Zahavi *et al.* 2009], only works in the absence of failures. However, we compare its performance with the linear-shift because the properties of this algorithm [Prisacari *et al.* 2013b] suggest it may offer better fault tolerance. Then, we discuss the inadequacies of current scheduling algorithms in avoiding congestion during failures, highlighting the necessity to increase the number of phases for the all-to-all exchange to effectively address failures. Finally, we introduce an algorithm and Integer Linear Programming (ILP) model that allows for the adaptation of any scheduling pattern to failures.

5.1 Latin Square

As introduced in 2.2.3, in the all-to-all collective exchange, each server exchanges data with all other servers through the network. A straight forward topology for the all-to-all exchange is a full-mesh network where each server is directly connected with all the other servers. This is however very costly and a waste of resources as not all links are needed all the time. This is why, the all-to-all collective exchange is typically divided into multiple phases to distribute the necessary bandwidth over time and avoid congestion in the network. The minimum number of phases necessary for the all-to-all exchange is n, the number of servers. At each phase, every server communicates exclusively with one other server that is not involved in any other communication to spread the load over the topology. The objective is for all servers to have communicated with each other by the end of all phases.

These constraints are the same as for Latin Square. A Latin square of order n is a $n \times n$ array filled with n different symbols, each occurring exactly once in each row and exactly once in each column. Latin squares were first designed by the mathematician Choi Seok-jeong in 1700, who used them in his work on constructing magic squares [Colbourn & Dinitz 2006].

We can define any all-to-all communication pattern using Latin squares of order n, where n is the number of servers. In this context, the source servers are represented by the rows, the phases by the columns, and the symbols within the Latin square represent the ID of the destination servers. For instance, we can define a linear-shift pattern [Zahavi *et al.* 2009] between four servers using a Latin square of order four as shown in Table 5.1. In this Latin square, each column is shifted by its column number starting from 0 with the permutation $[0, 1, 2, 3]$, meaning that each number d in a cell is computed by $d = (id_{row} + id_{column}) \mod 4$.

Table 5.1: Linear-shift pattern defined as a Latin Square.

	Phase 0	Phase 1	Phase 2	Phase 3
Server 0	0	1	2	3
Server 1	1	2	3	0
Server 2	2	3	0	1
Server 3	3	0	1	2

A Latin Square can also be represented as a $n^2 \times 3$ array, known as the orthogonal array representation[Stec 2023]. This representation includes three columns denoted r, c and s, where r is the row number in the $n \times n$ Latin Square, c is the column number and s is the cell value. The associated orthogonal array representation with the linear-shift pattern with 4 servers is shown in Table 5.2.

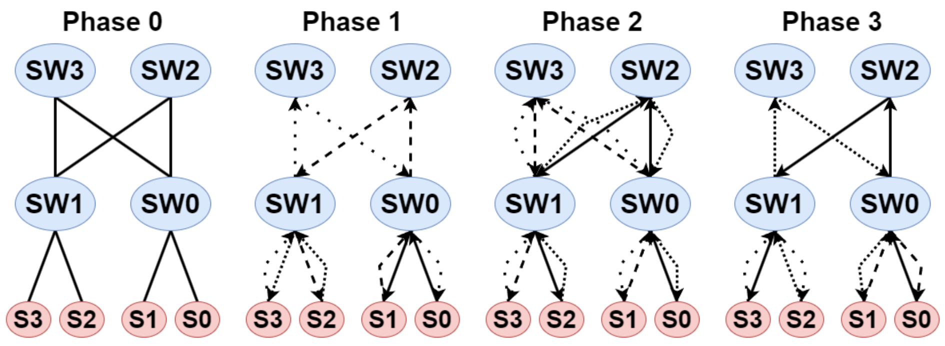
By observing the Latin Square in Table 5.1, we notice that the rows and columns are generated by a cycle of shifts. Specifically, rows are permutations of rows and columns are permutations of columns. Latin Squares can be described as a set of permutations where the sets of numbers $\{0, \ldots, n-1\}$ are rearranged according to the row and column. This implies that all sets of permutations that respect the properties of a $n \times n$ Latin Square are valid solutions for the all-to-all exchange, as they share the same properties. However, computing the number L_n of possible $n \times n$ Latin Squares remains challenging, as it is a complex combinatorial problem. While an exact enumeration of the solutions is possible for small n up to $n =$

Table 5.2: Linear-shift pattern defined as an orthogonal array representation of a Latin Square.

row	*column*	*symbol*
0	0	$s = (0+0) \mod 4 = 0$
0	1	$s = (0+1) \mod 4 = 1$
0	2	$s = (0+2) \mod 4 = 2$
0	3	$s = (0+3) \mod 4 = 3$
1	0	$s = (1+0) \mod 4 = 1$
1	1	$s = (1+1) \mod 4 = 2$
1	2	$s = (1+2) \mod 4 = 3$
1	3	$s = (1+3) \mod 4 = 0$
2	0	$s = (2+0) \mod 4 = 2$
2	1	$s = (2+1) \mod 4 = 3$
2	2	$s = (2+2) \mod 4 = 0$
2	3	$s = (2+3) \mod 4 = 1$
3	0	$s = (3+0) \mod 4 = 3$
3	1	$s = (3+1) \mod 4 = 0$
3	2	$s = (3+2) \mod 4 = 1$
3	3	$s = (3+3) \mod 4 = 2$

11, for larger n, researchers have derived both upper and lower bounds on the number of possible solutions, which differ significantly [Van Lint & Wilson 2001, McKay & Wanless 2005].

As we demonstrated, all $n \times n$ Latin Squares can satisfy the constraints of the all-to-all exchange scheduling between n servers. However, additional constraints are required as we also need for the schedule to map on the topology without congestion. This is exemplified by the linear-shift communication pattern, which is associated with the Ftree routing algorithm [Nvidia 2023b] to guarantee no congestion in the network under normal conditions. However, the linear-shift pattern shows significant congestion in the event of network failures, as demonstrated in Section 4.3.4. The bandwidth usage at the different phases of the linear-shift scheduling is not balanced. This leads to unavoidable congestion at certain phases as depicted in Figure 5.6. As explained in Section 2.3.2, each communication flow uses all the bandwidth of a link. For example, Figure 5.1 illustrates the Ftree routing of linear-shift scheduling with four servers. At phase 0, there is no congestion because each server exchanges data with itself, resulting in no link usage. However, at phase 2, all links are used in both directions, leading to inevitable congestion in the event of failures. To address this issue, our initial approach was to identify a more fault-tolerant scheduling pattern that could better handle network failures compared to the linear-shift.



Server 0	0	1	2	3
Server 1	1	2	3	0
Server 2	2	3	0	1
Server 3	3	0	1	2

Figure 5.1: The Ftree routing of the Linear-shift communication pattern at each phase. The topology of the network is a two-layer fat-tree denoted $FT(2;2,2)$, which interconnects four servers. $SW[0,\ldots,3]$ are the switches and $S[0,\ldots,3]$ are the servers. The table is the same as Table 5.1, which shows the scheduling of the linear-shift defined as a Latin square. The first column is the source server, each following column is a phase and the content in the cell is the destination server.

5.2 Bandwidth reduction

Before discussing an alternative all-to-all scheduling approach that could potentially better address failures, it is important to introduce a key concept, which is one of our contributions: bandwidth reduction. Bandwidth reduction allows us to more precisely define the impact of failures on the routing of an all-to-all schedule. We define bandwidth reduction as the maximum number of failures occurring on the same leaf switch. In this chapter, when we use the term bandwidth reduction, we always refer to a uniform bandwidth reduction that results in the same number of paths between a leaf switch affected by a failure and all other leaf switches. For instance, the bandwidth reduction of the top failure scenario in Figure 5.2 is uniform as the leaf switch SW0 has the same number of paths, which is 3, to join the other leaf switches through SW5..7. A non-uniform bandwidth reduction would be if there is an additional failure between the switches SW1 and SW5. In that case, SW0 and SW1 have only two paths to join each other (through SW6 and SW7) while having three paths to join the other leaf switches. The concept of non-uniform bandwidth reduction will be discussed with more details in Section 6.2.

In Figure 5.2, we illustrate the fat-tree topology FT(2;4,4) as an example. In the top figure, we show the traffic to and from servers S0..3 using different types of arrows. We consider the worst-case scenario for bandwidth usage in the scheduling, where servers S0..3 must communicate externally and none communicate locally (meaning none of the servers S0..3 are communicating with servers S0..3). The

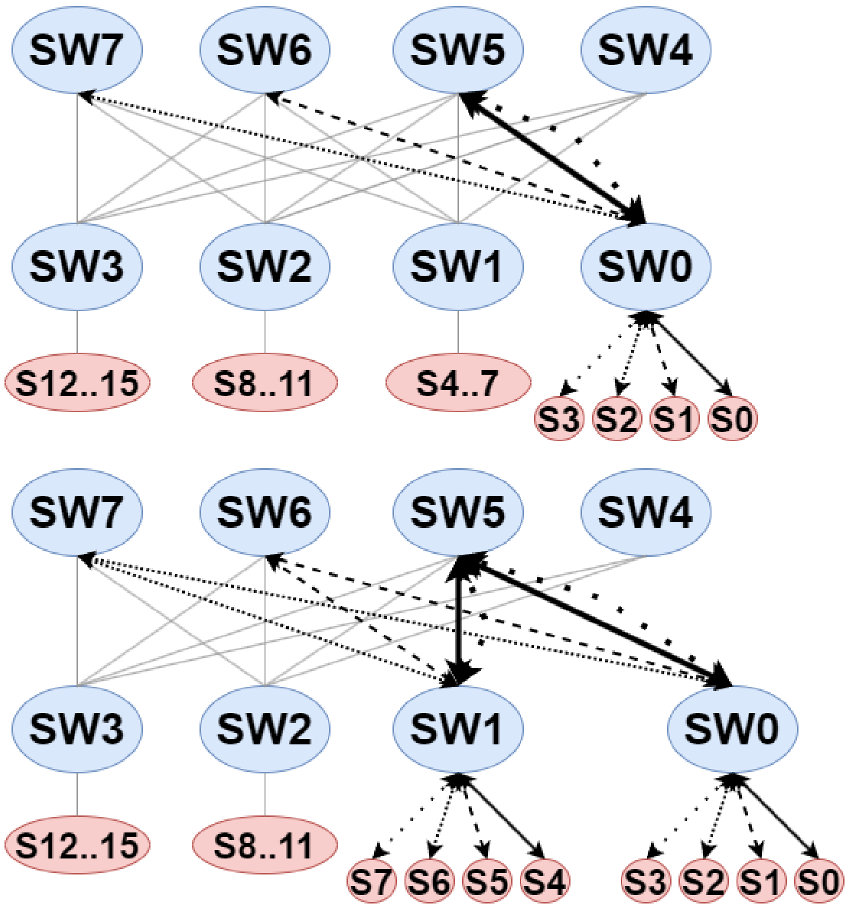


Figure 5.2: An example of the impact of a bandwidth reduction of 1 in the FT(2;4,4) topology with 16 servers. The different types of arrows represent the traffic in both directions towards the associated server. In the top figure, the bandwidth reduction is 1 due to a link failure between SW0 and SW4. As a result, servers S0..3 have only three links available for external communication, creating one congestion on the link between switches SW0 and SW4. Similarly, the bottom figure illustrates a bandwidth reduction of 1 with two failures, one between SW0 and SW4 and another between SW1 and SW4, resulting in one congestion on the links SW0-SW5 and SW1-SW5.

failure between switches SW0 and SW4 reduces the bandwidth by 1, resulting in servers S0..3 having only 3 links instead of 4 for simultaneous communication. The bandwidth reduction is uniform because the leaf switch affected by the failure, SW0, has three available paths to all other leaf switches, through the spine switches SW5 to SW7. If we distribute the congestion across the links as evenly as possible, this means that the links between switches SW0 and SW5 is used twice in the same direction, creating one congestion on that link.

This congestion means that the communications to and from servers S0 and S3 will take twice as long to complete. Consequently, since the all-to-all exchange is synchronized, all other servers will have to wait for S0 and S3 to finish their exchanges before moving on to the next phase. If we assume that this phase normally takes time T, due to this congestion, it will take a duration of $2T$. Since the exchange is synchronized, all other servers will be impacted by the failure.

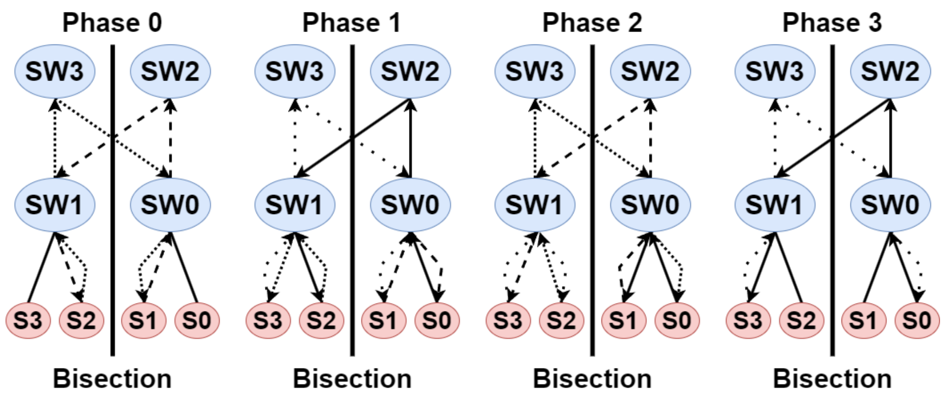
In the bottom sub-figure of Figure 5.2, the bandwidth reduction is still 1, but in this case there are two failures: one between switches SW0 and SW4 and another between switches SW1 and SW4. Similarly to the previous case, if the load is optimally distributed between the links, this failure scenario creates congestion on the links SW0-SW5 and SW1-SW5. The bandwidth reduction is also uniform as the leaf switches SW0 and SW1 affected by the failures have three available paths to join all other leaf switches, through the spine switches SW5 to SW7. In this scenario, in the same manner as before, the communications to and from servers S0 and S3, as well as S4 and S7, will take twice as long to complete. This affects the time required to finish this phase, as it will also take 2T instead of T.

Therefore, it does not matter if there is a failure between switches SW0 and SW4 or if the spine SW4 is completely disconnected, since the maximum number of congestion on the links will be the same (if the routing is optimal), affecting the time to complete the phase in the same way. That is why, in this chapter, we prefer to evaluate our solution in terms of bandwidth reduction rather than the number of failures.

5.3 The Bandwidth-Optimal All-to-All exchange

The bandwidth-optimal all-to-all exchange is described in [Prisacari *et al.* 2013b]. The authors present an all-to-all scheduling pattern specifically adapted to fat-tree topologies. This pattern optimizes bandwidth usage at each phase. The authors compare their scheduling pattern with linear-shift [Zahavi *et al.* 2009] and XOR, demonstrating that their approach minimizes the number of messages crossing the bisection bandwidth.

Figure 5.3 illustrates an example of the bandwidth-optimal all-to-all communication pattern of Prisacari et al. [Prisacari *et al.* 2013b] with four servers. At each phase, exactly 2 messages cross the bisection bandwidth, corresponding to half the messages sent during a phase. This contrasts with the linear-shift where there is increasing use and then decreasing use of the bisection bandwidth. In Figure 5.1, the number of messages crossing the bisection is 0, 2, 4, and 2 for phases 0, 1, 2, and 3 respectively. Prisacari et al. prove that half of the messages crossing the bisection at each phase is optimal. This scheduling solution was our initial approach, as the balanced bandwidth usage at each phase would allow better adaptation in the event of failures compared to the linear-shift. However, a challenge we encountered was that the authors did not provide a well-defined algorithm for their solution, only a set of formulas for computing the destination depending on the source and the phase and a proof that this satisfies the properties of the all-to-all exchange. Therefore, we first propose Algorithm 1 which is a well-defined algorithm to compute the



Server 0	0	2	1	3
Server 1	2	0	3	1
Server 2	1	3	0	2
Server 3	3	1	2	0

Figure 5.3: The routing of the Bandwidth-optimal communication pattern at each phase. The topology of the network is a two-layer fat-tree denoted $FT(2;2,2)$, which interconnects four servers. $SW[0,\dots,3]$ are the switches and $S[0,\dots,3]$ are the servers. At each phase, only two messages cross the bisection of the network.

all-to-all exchange scheduling on a fat-tree topology. The complexity of Algorithm 1 is $O(S^2 \times L)$ where S is the number of servers and L is the number of layers in the fat-tree topology.

The notations used in the algorithm are consistent with those presented in [Prisacari *et al.* 2013b] and are summarized in Table 5.3. The concept of a variable base is defined in [Prisacari *et al.* 2013b]. In a fixed-base system, the base remains constant. For instance, in base 2, each digit can only take the value 0 or 1. In a variable base, however, the base can change dynamically for different positions within the number. In the bandwidth-optimal scheduling, each destination server is given by computing each digit using the base specific to each layer in the fat-tree.

Table 5.3: Notations of Algorithm 1.

Name of the variable	Description of the variable
s	The source server.
d	The destination server.
p	The phase.
S	The number of servers.
P	The number of phases which is equal to S.
L	The number of layers in the fat-tree topology.
$M = [M_0, \ldots, M_{L-1}]$	The list M where each element M_l in the list represents the number of links connected to a switch at layer $l \in [0, \ldots, L-1]$, with 0 being the bottom layer connected to the servers and $L-1$ being the top layer.
$kPerm$	The list of size L obtained by reversing the order of the elements in $[1, \ldots, L]$.
α	The digits of the destination server d in the variable base $(M_0, \ldots, M_{L-1})$.

After developing a well-defined algorithm that computes the destination according to the source and phase to reproduce the bandwidth-optimal scheduling defined in [Prisacari *et al.* 2013b], we compare the congestion produced by the bandwidth-optimal scheduling in the event of failures with the congestion produced by the linear-shift scheduling.

To simulate the congestion of the linear-shift and bandwidth-optimal, we use a simple algorithm to compute the number of conflicts at each phase and the resulting duration of the phase according to the failure scenario, as described in Algorithm 2. Given the algorithmic nature of these patterns, a known topology and failure scenario, predicting the number of conflicts is straightforward.

First, we compute the communication pairs for each phase and for each scheduling pattern. Then, we simulate failure scenarios and compute the number of external communications to and from the leaf switches, as well as the available bandwidth for each leaf switch. This allows us to identify communications that share a link, as we know which server is connected to a leaf switch with a broken link and the communications to these servers. We define an external communication as one occurring between a source server and a destination server not connected to the same switch. This type of communication requires using links that ascend to the top

Algorithm 1: Computation of the Bandwidth-Optimal scheduling.

```
1  Input: Number of servers and phases $S$
2  Input: Number of layers $L \geq 2$
3  Input: The list $M$ that contains the number of links connected to each
     switch at layer $l \in [0, \dots, L-1]$
4  Output: The dictionary $schedule$ with the phase $p$ as key and the list of
     tuple $(s, d)$ as value, where $s$ is the source server and $d$ is the destination
     server
5  $M_0$ := First element of the list $M$ and the number of servers connected to
     each leaf switch
6  $M_1$ := Last element of the list $M$ and the number of leaf switches
     connected to each spine switch
7  $kPerm = [L, L-1, \dots, 2, 1]$
8  $schedule$ := Empty dictionary of size $S$
9  forall $p \in [0, \dots, S-1]$ do
10     $schedule[p]$ := Empty array of size $S$
11     forall $s \in [0, \dots, S-1]$ do
12         $d = 0$
           // The $\alpha$ are the values of the destination at each layer
13         $\alpha$ := Empty array of size $L$
14         forall $k \in [0, \dots, L-1]$ do
               // Determining the factor to compute $\alpha[k]$
15             if $kPerm[k] - 1 == 0$ then
16                 $factor = 1$
17             else if $kPerm[k] - 1 == 1$ then
18                 $factor = M_1$
19             else
20                 $factor = M_1 \times (kPerm[k] - 2) * M_0$
21             $\alpha[k] = \left\lfloor \frac{s}{factor} \right\rfloor + \left\lfloor \frac{p}{factor} \right\rfloor \mod M[kPerm[k] - 1]$
22         forall $i \in [0, \dots, L-1]$ do
23             $d = d + \alpha[i] \times (M_0)^i$
24         $schedule[p].insert((s, d))$
25 return $schedule$
```

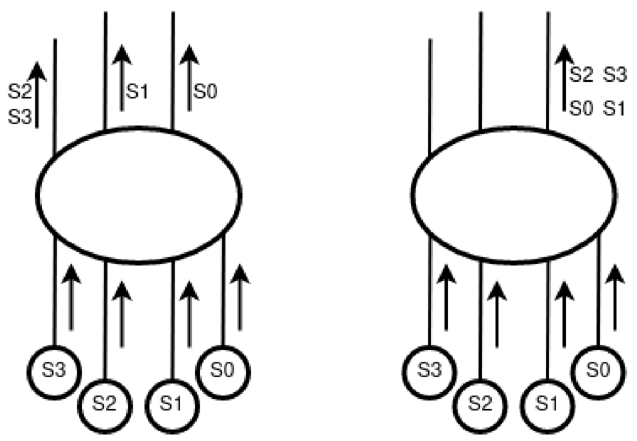


Figure 5.4: The Min-Hop routing algorithm's best-case scenario (left) and worst-case scenario (right) in terms of congestion distribution.

layer in the fat-tree topology and then descend to the bottom layer to reach the destination server.

In Algorithm 2, if no communication flows use the links between the leaf and spine switches, *i.e.*, no server communicates with other servers, the phase is almost instantaneous. If for each leaf the number of communication flows to or from the leaf during a phase is less than or equal to the number of available links connected to the leaf, there is sufficient bandwidth. In this case, there will be no conflicts, and the communications from or to the leaf switch takes 1 unit of time. Otherwise, we assume the communications are evenly distributed across the available links, with the phase duration computed by the maximum number of communications sharing a link which is the sharing factor. The duration of a phase D_p is computed using the formula:

$$D_p = \max\left(\left\lceil \frac{C_l}{N_l} \right\rceil \mid l \in L\right),$$

where L is the set of leaf switches, C_l represents the number of external communications from or to the leaf $l \in L$, and N_l is the number of available links connected to leaf $l \in L$. As the all-to-all exchange is synchronized, all communications during a phase must wait for each other to finish before moving to the next phase. This means that if a communication between two servers takes X units of time, and X is the maximum duration of any exchange during the phase, then the duration of the phase is X. To compute the duration of the all-to-all exchange D, we simply need to sum the duration of all phases, which is given by this formula:

$$D = \Sigma_{p=0}^{p=P-1} D_p$$

where P is the number of phases.

For the routing, we consider the best-case in terms of distribution of the congestion for the Min-Hop routing algorithm, which is the default routing algorithm used by the OpenSM controller in the event of a failure[Nvidia 2023b]. We assume

Algorithm 2: Computation of the duration of the all-to-all schedule.

```
1  Input: The dictionary $schedule$ with the phase as key and the ordered list
     of source-destination pairs $(s, d)$ as value
2  Input: The ordered list $L$ of leaf switches
3  Input: The dictionary $fromLeaf$ with the phase $p \in P$ and leaf switch
     $l \in L$ as key and the ordered list of tuple $(s, d)$ as value, where $s$ is the
     source server connected to $l$ and $d$ is the destination server
4  Input: The dictionary $toLeaf$ with the phase $p \in P$ and leaf switch $l \in L$
     as key and the ordered list of tuple $(s, d)$ as value, where $s$ is the source
     server and $d$ is the destination server connected to $l$
5  Input: The dictionary $bw$ with the leaf switch $l \in L$ as key and the
     bandwidth available (number of links) for the leaf $l$ as value
6  Input: The number of servers $M_0$ connected to the leaf switches
7  Output: The total duration of the all-to-all schedule $D$
8  $D = 0$
9  forall $p \in schedule$ do
       // $D_p$ is the duration of the phase
10     $D_p = 0$
11     forall $c \in schedule[p]$ do
12         $s = c[0]$
13         $d = c[1]$
           // Computation of the source leaf switch $sLeaf$ and the destination
              leaf switch $dLeaf$ with the source server $s$ and the destination
              server $d$
14         $sLeaf = \left\lfloor \frac{s}{M_0} \right\rfloor$
15         $dLeaf = \left\lfloor \frac{d}{M_0} \right\rfloor$
16         if $sLeaf \neq dLeaf$ then
               // If at least one communication $(s, d)$ needs to use the links
                  between the leaf and spine switches, the duration of the phase
                  is 1
17             $D_p = 1$
18             break
19     forall $l \in L$ do
20         $t = \left\lfloor \frac{|fromLeaf[(p,l)]|}{bw[l]} \right\rfloor$
21         if $t > D_p$ then
22             $D_p = t$
23         $t = \left\lfloor \frac{|toLeaf[(p,l)]|}{bw[l]} \right\rfloor$
24         if $t > D_p$ then
25             $D_p = t$
26     $D = D + D_p$
27 return $D$
```

that it will spread the routes as efficiently as possible in case of failures, although this is not always the case in practice. An example of the best-case and worst-case scenarios for the Min-Hop algorithm is shown in Figure 5.4. In this example, four servers need to communicate externally while only three links are available on the switch. In the best-case scenario (left), there is only one conflict. In the worst-case scenario (right), there are three conflicts. We only use the best-case scenario of Min-Hop to evaluate the bandwidth-optimal and linear-shift scheduling as it is the most optimized one.

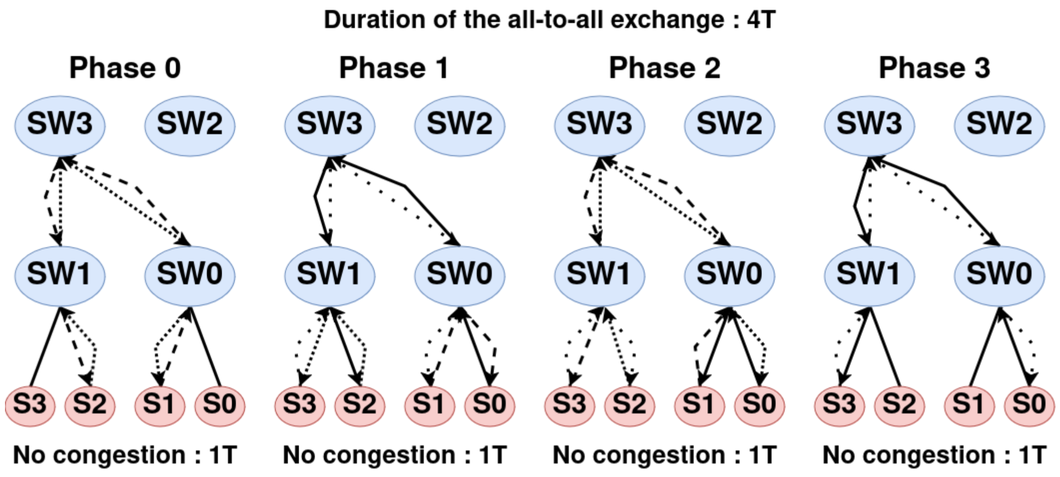


Figure 5.5: The routing of the Bandwidth-optimal communication pattern at each phase with one link failure between SW0 and SW2. The topology of the network is a two-layer fat-tree $FT(2;2,2)$, which interconnects four servers. $SW[0,\ldots,3]$ are the switches and $S[0,\ldots,3]$ are the servers. Each phase takes one unit of time as there is still enough bandwidth to route the exchanges at each phase. The total duration of the all-to-all exchange in this case is of four unit of time.

In order to give a clear example of this algorithm, we can consider the example of the routing of the bandwidth-optimal scheduling in Figure 5.3. Figure 5.5 illustrates the routing of the bandwidth-optimal scheduling when the link between switches SW0 and SW2 is broken. With bandwidth-optimal scheduling, a single failure in this topology poses no problem, as there is always sufficient bandwidth to route the exchanges at each phase. At each phase, at most one communication between a source server and a destination server needs to use the links between the bottom and top layers in the fat-tree topology (the links between SW[0,1] and SW[2,3]) and there is always at least one available link. All links have the same capacity and can be used in both directions. Consequently, each phase takes one unit of time.

To give a clear example of the computation of an all-to-all exchange with Algorithm 2, Figure 5.6 illustrates the routing of linear-shift scheduling with one link failure. We consider phase 0 to be instantaneous because each server communicates with itself, which is almost instantaneous in practice. Phases 1 and 3 each take one unit of time as there is no congestion. Phase 2, however, creates congestion between

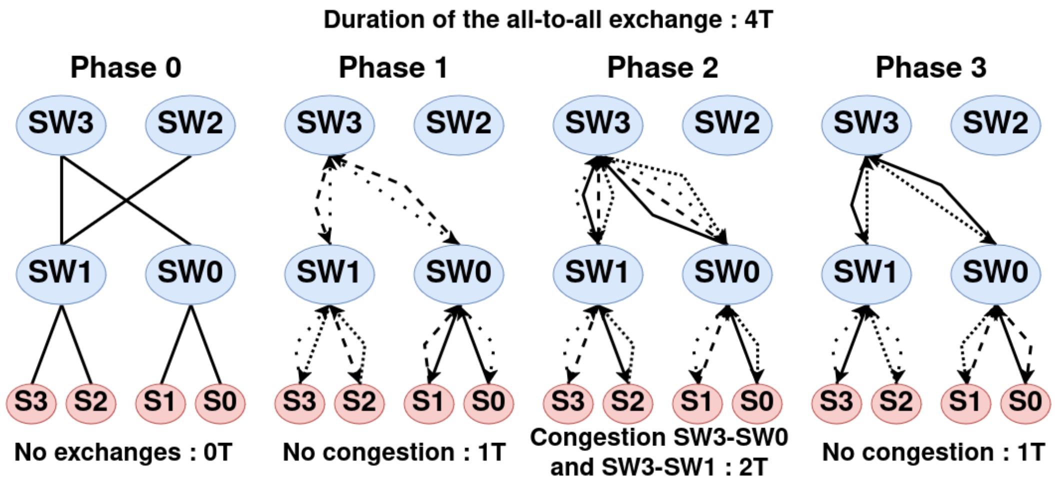


Figure 5.6: The routing of the linear-shift communication pattern at each phase with one link failure between SW0 and SW2. The topology of the network is a two-layer fat-tree denoted $FT(2;2,2)$, which interconnects four servers. $SW[0,\dots,3]$ are the switches and $S[0,\dots,3]$ are the servers. Phase 0 is almost instantaneous because each server communicates with itself. Phases 1 and 3 each take one unit of time since there is sufficient bandwidth to handle the exchanges. However, congestion occurs at phase 2 as two communications must share two links. Consequently, the total duration of the all-to-all exchange in this scenario is four units of time.

SW0-SW3 and SW3-SW1. For example, communications between S2-S0 and S3-S1 need to share the links SW1-SW3 and SW3-SW0 in the same direction. Since two communications are using the same link, it takes twice as long to route this exchange. Therefore, the duration of phase 2 is 2T. Despite this, the total duration of the all-to-all exchange with the linear-shift scheduling is equal to the duration of the all-to-all exchange with bandwidth-optimal scheduling. However, this will not necessarily be the case for larger topologies, as we show in Figure 5.7.

In Figure 5.7, we show the simulated time performance of the linear-shift and bandwidth-optimal scheduling under bandwidth reduction using the topology of the LHCb DAQ network, which is our case of study. The failures occur on one leaf switch. However, the result is the same whether the failures occur on one leaf switch or all the leaf switches. Since the exchange is synchronized, the bandwidth reduction effect is the same whether there is one failure on one leaf switch or a failure with the same spine on each leaf switch in the fat-tree topology, as the servers need to wait for each other to move to the next phase.

This is why we prefer to use the term "bandwidth reduction" instead of "failures". In this simulation, the specific leaf switch on which the failures occur does not matter as much as the overall bandwidth reduction. Therefore, when the bandwidth reduction is indicated as 1 on the x-axis, it does not imply a single failure in the entire topology; rather, it implies at least one failure on at least one leaf switch, potentially affecting all leaf switches as explained in Section 5.2. This could

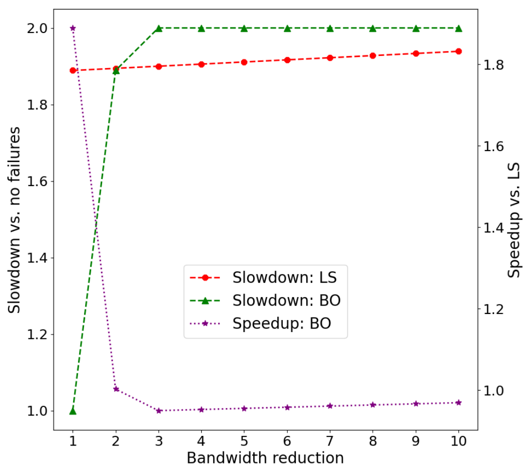


Figure 5.7: Time performance with the topology FT(2;20,18), which is the topology of the LHCb DAQ network. LS refers to the linear-shift scheduling and BO is the bandwidth-optimal scheduling. The left axis shows the slowdown of the different scheduling upon bandwidth reduction compared to no failures, this is the relative time degradation due to failures. The speedup, on the right axis, shows the improvement ratio of bandwidth-optimal compared the linear-shift. The failures happen on the leaf switches, which are the links between the bottom and top layer in the fat-tree topology.

correspond to 1 to 20 failures, as there are 20 leaf switches in the fat-tree topology FT(2;20,18). If there are two failures on the same leaf switch, the bandwidth reduction is 2, and the duration of the all-to-all exchange remains the same, regardless of whether there are two failures on one leaf switch or two failures on each of the 20 leaf switches (resulting in 40 failures).

A bandwidth reduction of 10 in the topology $FT(2;20,18)$ can result in the loss of half of the links in the network. This is our most extreme failure scenario as more failures are not realistic and cannot be optimized since too much bandwidth is lost. 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. These failure scenarios do not result in a reduction of bandwidth that may lead to congestion as the data to transmit reduces by the same amount of the bandwidth in the topology.

We define the *slowdown* of a scheduling solution in 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, the latter being the ideal performance. The time it takes to complete the all-to-all exchange is on the order of tens or hundreds of milliseconds, depending on the number of servers, as each phase takes approximately 2 milliseconds. However, the unit of time does not matter for the slowdown since it is a ratio. Basically, for every failure scenario, we compute the slowdown with this formula:

$$\text{Slowdown} = \frac{T}{S}$$

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 number of phases in an all-to-all schedule without failures.

We define the *speedup* as the improvement ratio of the bandwidth-optimal compared to the linear-shift scheduling. We compute the speedup with this formula:

$$\text{Speedup} = \frac{T_{LS}}{T_{BO}}$$

With a bandwidth reduction of 1, the bandwidth-optimal compared to the linear-shift scheduling shows a speedup factor of two. This is due to the fact that with one failure, the bandwidth-optimal scheduling has no conflicts at each phase as there are fewer external communications that cross the bisection bandwidth at each phase. However, when the bandwidth reduction is greater or equal to 2, the linear-shift has slightly better performance than the bandwidth-optimal scheduling. During the initial and final phases in the linear-shift scheduling, servers communicate with other servers connected to the same leaf switch (in the first phase all servers communicate with themselves). Therefore, in the initial and final phases, almost no bandwidth is used which results in the linear-shift handling the congestion better than the bandwidth-optimal scheduling that has congestion at every phase because there are always external communications at every phase. Even if the middle phases in the linear-shift scheduling result in a lot of congestion as all servers talk externally, this cost is compensated by the initial and final phases. However, as shown in Figure 5.4, we only took into account the best-case of the routing algorithm Min-Hop, which favors the linear-shift scheduling as the congestion is optimally distributed. If we do not use the best case of Min-Hop, which could be the case in practice, the bandwidth-optimal scheduling could perform better than the linear-shift because the number of conflicts with the linear-shift scheduling would increase if the external communications are not well distributed on the links.

Except for a bandwidth reduction of one, the speedup of the bandwidth-optimal scheduling compared to the linear-shift scheduling does not show much improvement as the ratio is 0.9 and is closer to 1 when the bandwidth reduction increases.

In any case, it is clear that with these two scheduling solutions, congestion is unavoidable at some point and the duration of the all-to-all is generally increased by a factor close to 2 with the two scheduling solutions, as indicated by the slowdown ratio compared to the scenario with no failures. Unfortunately, we could not find in the literature a scheduling algorithm that adapts better to failure, which is why we decided to adapt the existing ones to address them.

5.4 Adaptation of the Bandwidth All-to-All Exchange to Failures

In this section, we introduce an initial approach that adapts the bandwidth-optimal scheduling to address failure scenarios effectively. Specifically, it must produce a

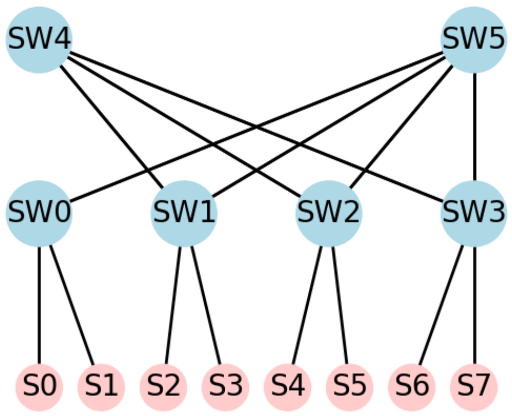


Figure 5.8: The fat-tree topology $FT(2;2,4)$ with 6 switches and 8 servers. The link between SW0 and SW4 is broken so it is not represented in the topology.

schedule that ensures a routing solution is possible without congestion on the links. Our first contribution demonstrates that the number of phases, S, for an all-to-all exchange between S servers is insufficient for achieving a congestion-free exchange. Consequently, we propose increasing the number of phases and provide a lower bound on this number to minimize the total exchange duration. Then, we propose an algorithm and an Integer Linear Programming (ILP) model to adapt the bandwidth-optimal scheduling within this minimum number of phases.

5.4.1 Lower bound on the number of phases with bandwidth reduction

In Section 5.3, we observed that current scheduling algorithms do not effectively address the problem of bandwidth reduction. All existing scheduling approaches lead to congestion because the number of phases is insufficient to achieve an all-to-all exchange with bandwidth reduction. The scheduling algorithms we reviewed provide a scheduling in S phases for the all-to-all exchange between S servers, making them inadequate for adapting to failures.

To illustrate, consider the fat-tree topology in Figure 5.8 with a failure between switches SW0 and SW4. In this scenario, servers S0 and S1 are unable to send or receive data simultaneously as there is only one link available (SW0-SW5). Consequently, they must send or receive data sequentially. Referring to the Latin Square of the bandwidth-optimal scheduling in Table 5.4, some communications in specific phases must be adjusted to prevent congestion. For instance, at phase 1, communications S0-S2 and S1-S4 create congestion on the link between SW0 and SW5 since both servers connected to SW0 are communicating externally. Therefore, we should remove either the communication S0-S2 or S1-S4 and reschedule one of them to another phase to avoid congestion. In the example provided in Table 5.4, eight communications are moved to the additional phases $P[8..11]$, ensuring a congestion-free schedule.

Table 5.4: The bandwidth-optimal pattern is defined as a Latin Square with a bandwidth reduction of 1. To prevent congestion, destination servers marked with the symbol "*" need to be removed from the original phases and moved to the additional phases $P[8..11]$.

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
S0	S0	S2*	S4*	S6	S1	S3*	S5*	S7	S2	S4	S3	S5
S1	S2	S4	S6	S0	S3	S5	S7	S1				
S2	S4	S6	S0*	S2	S5	S7	S1*	S3	S0	S1		
S3	S6	S0*	S2	S4	S7	S1*	S3	S5		S0	S1	
S4	S1	S3	S5	S7	S0	S2	S4	S6				
S5	S3	S5	S7	S1	S2	S4	S6	S0				
S6	S5	S7	S1	S3	S4	S6	S0	S2				
S7	S7	S1	S3	S5	S6	S0	S2	S4				

This approach ensures congestion-free scheduling when combined with appropriate routing. However, adding phases increases the duration of the all-to-all exchange due to the additional time required for these phases and their synchronization. Therefore, we propose a lower bound on the number of phases according to the bandwidth reduction in a fat-tree topology to ensure congestion-free scheduling while minimizing the duration of the all-to-all exchange.

Our lower bound is true for any logical fat-tree $LFT(L; M_1, \ldots, M_L)$ and generalized fat-tree $FT(L; M_1, \ldots, M_L)$. The example topology shown in Figure 5.8 has two layers and 8 servers and is denoted $FT(2; 2, 4)$. Let $\Pi_l = \prod_{i=1}^{l} M_i$ represents the bandwidth at layer l. Π_L then represents the bandwidth in the whole fat tree as well as the number of servers. Here, $\Pi_L = M_1 * M_2 = 2 * 4 = 8$.

In this section, we call *local* servers of a specific switch the servers that belong to the sub-tree below the switch. The other servers are called *external*. For example, the number of external servers for each leaf switch in Figure 5.8 is 6. If a server local to a switch communicates with an external server, this communication will go through the links connecting the local switch to its parent at layer $l + 1$. In the all-to-all exchange, the number of communications of local servers with external servers is therefore given by:

$$U_l = \Pi_l(\Pi_L - \Pi_l) = 2 * (8 - 2) = 12$$

which is, in this example, 12. To better explain this result, if we look at, for example, switch SW0 we can see that it has two local servers connected to it. Each of these two servers communicate once with the 6 external servers in an all-to-all collective exchange. Therefore, the switch SW0 exchanges $2 * 6 = 12$ messages with the other switches in the topology.

Now we consider that the link capacity between a switch SWX with $X \in 0..5$ at layer l and a switch at layer $l + 1$ is reduced by F, with $1 \leq l \leq L - 1$ and $F < \Pi_l$. We must divide this number of messages exchanged by the number of links still available on at least one switch. Hence, the minimum number of phases to send all these messages if the link between SW0 and SW4 is not broken in the topology is:

$$\left\lceil \frac{U_l}{\Pi_l - F} \right\rceil$$

$$\left\lceil \frac{12}{2-0} \right\rceil = 6$$

However, if one of the leaf switches loses one of its links connected to its parents at layer $l+1$, the two servers connected to it have only one link to communicate with the external servers. For example, if the link between SW0 and SW4 is broken, the servers S0 and S1 need to communicate in turns with the external servers, which means that switch SW0 needs $\left\lceil \frac{12}{2-1} \right\rceil = 12$ phases to execute its full external exchange. The other switches without failures still need only $\left\lceil \frac{12}{2-0} \right\rceil = 6$ phases to do their full external exchanges since they have both of their links available. Therefore, switch SW0 is the switch that affects the number of phases needed to complete the all-to-all exchange. That is why the value of F in the computation of the lower bound should be the maximum number of failures on a same switch, *i.e.*, the bandwidth reduction. Also, we know that the number of phases is at least the number of servers Π_L. Thus, the number of phases necessary to perform the all-to-all exchange verifies

$$P \geq \max\left(\Pi_L, \left\lceil \frac{U_l}{\Pi_l - F} \right\rceil\right) \tag{5.1}$$

We use the maximum between these two values because in some topologies we do not necessarily need to increase the number of phases. Interestingly, using the same reasoning, the lower bound remains the same even if each switch at layer l has F links failure with their parents at layer $l+1$. Similarly, if the capacity of each link between layer $l+1$ and $l+2$ is reduced by $F \times M_{l+1}$, then the lower bound remains unchanged because even with a bandwidth of $\Pi_l M_{l+1} - F M_{l+1} = M_{l+1}(\Pi_l - F)$, it is equal to the sum of the bandwidth at the layer just below. In fact, not only the lower bound is the same, but a schedule that tolerates reduction in capacity of F between layer l and $l+1$ also tolerates a reduction of capacity of $F \times M_{l+1}$ between layer $l+1$ and $l+2$. This is explained by the concept of bandwidth reduction detailed in Section 5.2.

5.4.2 Increase in the number of phases

Figure 5.9 illustrates the multiplicative factor I for the number of phases in the fat-tree topology, computed with 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

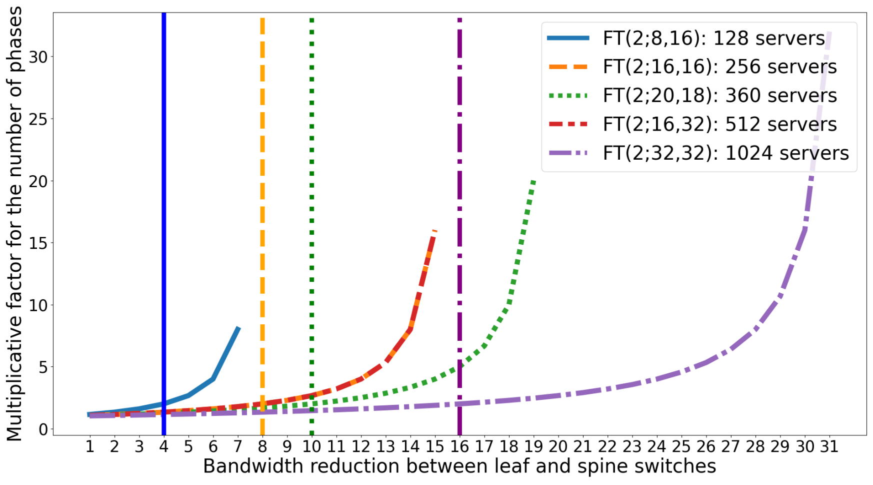


Figure 5.9: Multiplicative factor for the number of phases according to the bandwidth reduction f between leaf and spine switches for several fat-tree topologies. FT(2;8,16) contains 128 servers, FT(2;16,16) contains 256 servers, FT(2;20,18), our use-case network, contains 360 servers, FT(2;16,32) contains 512 servers and FT(2;32,32) contains 1024 servers. The vertical lines correspond to the loss of half the bandwidth for each topology.

the maximum number of link failures on a same leaf switch. 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 5.4.1, 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 reduction between the leaf and spine switches, P is the number of servers and M_0 is the number of servers connected to each leaf switch. Consequently, as the bandwidth reduction increases, the throughput decreases, but the decline is less pronounced compared to other solutions. 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. 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 a bandwidth reduction $f = \frac{M_0}{2} = \frac{20}{2} = 10$ 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 is lost in the network.

5.4.3 Scheduling the communications on the added phases

After determining the optimal number of phases to address bandwidth reduction, our next contribution is to schedule the communication accordingly to avoid congestion. Our initial approach involves adapting the bandwidth-optimal schedule by adjusting the placement of communications that could cause congestion in the additional phases. The bandwidth-optimal all-to-all pattern allows for the use of only half of the full bisection bandwidth. Consequently, the number of used links is more balanced between the different phases compared to the linear-shift. While the proposal in [Prisacari *et al.* 2013b] is appealing, no routing algorithm has been proposed to complement the schedule. Furthermore, it provides rules to be followed by the schedule but does not consider the necessary adaptations to the schedule and the routing upon link failures, as explained in Section 5.3. We propose an extended version of this algorithm that is more fault-tolerant.

As the number of phases increases, the problem becomes a variant of a partial Latin rectangle that is more difficult to solve. A partial Latin rectangle is defined as an $r \times c$ array where each cell is either empty or contains a symbol $n \in N$. Each symbol can appear at most once in each row and column [Falcón 2015].

In our problem, the number of columns c is greater than or equal to the number of rows r, with the rows representing the source servers and the columns representing the phases. In a Latin square, $r = c$, as the number of phases required for an all-to-all exchange is equivalent to the number of servers. However, to meet bandwidth reduction requirements, the number of phases must increase, leading to $r \leq c$. The presence of empty cells in a partial Latin rectangle limits the choices available for filling the remaining cells, which increases the complexity of the problem.

Another particularity of our problem compared to the standard partial Latin rectangle is that the number of symbols $|N|$ is equal to the number of rows r, since the rows correspond to source servers and the symbols represent destination servers. Additionally, each symbol must appear exactly once in each row, ensuring that each source server communicates exactly once with each destination server.

To adapt the bandwidth-optimal schedule, we solve our problem in two steps. First, we compute the communications to move to the additional phases by using Algorithm 3. Then, we assign theses communications to the phases. The complexity of Algorithm 3 is $O(|P| * |L|)$, where $|P|$ represents the number of phases and L is the number of leaf switches. We consider a reduction in capacity of $F \leq M_0 - 1$ between two switches, meaning at least one link stays up for each leaf in the corresponding fat-tree. M_0 is the number of servers directly connected to a leaf switch. Note that a conflict-free scheduling and routing occurs when at most one source-destination pair uses a link at a given time in the fat-tree.

Algorithm 3 identifies the communications crossing the failed links. It ensures that, on a switch with F failed links, there are no more than $M_1 - F$ incoming and outgoing communications to prevent conflicts on the links. Communications on links not affected by the failures are left untouched as they can be routed without conflicts.

Algorithm 3: Computation of the communications that need to be moved to the additional phases.

```
1  Input: The ordered list $P$ of phases
2  Input: The ordered list $L$ of leaf switches
3  Input: The dictionary $fromLeaf$ with the phase $p \in P$ and leaf switch
      $l \in L$ as key and the ordered list of tuple $(s, d)$ as value, where $s$ is the
      source server connected to $l$ and $d$ is the destination server
4  Input: The dictionary $toLeaf$ with the phase $p \in P$ and leaf switch $l \in L$
      as key and the ordered list of tuple $(s, d)$ as value, where $s$ is the source
      server and $d$ is the destination server connected to $l$
5  Input: The dictionary $bw$ with the leaf switch $l \in L$ as key and the
      bandwidth available (number of links) for the leaf $l$ as value
6  Output: The list of communications $cToMove$ that needs to be reschedule
      to the additional phases
7  forall $p \in P$ do
8      forall $l \in L$ do
           // If the number of external communications from leaf $l$ at phase $p$ is
              superior to the number of links connected to $l$, we need to
              reschedule the communications from leaf $l$ at phase $p$.
9          if $|fromLeaf[(p, l)]| > bw[l]$ then
10             forall $c \in fromLeaf[(p, l)]$ do
11                 if $c \notin cToMove$ then
12                     $cToMove.insert(c)$
           // If the number of external communications to leaf $l$ at phase $p$ is
              superior to the number of links connected to $l$, we need to
              reschedule the communications to leaf $l$ at phase $p$.
13         if $|toLeaf[(p, l)]| > bw[l]$ then
14             forall $c \in toLeaf[(p, l)]$ do
15                 if $c \notin cToMove$ then
16                     $cToMove.insert(c)$
17 return $cToMove$
```

Once we have the list of communications to move, we set up an integer linear programming (ILP) model with the Gurobi optimizer [experts 2023a] to schedule the communications computed in Algorithm 3 to the additional phases. Our choice of Gurobi is guided by its improved performance compared to other widely used solvers like IBM CPLEX and lpSolve [Luppold *et al.* 2018].

We were unable to find a general algorithm for scheduling communications to additional phases, as the communications that create congestion depend on the specific failure scenario. Such an algorithm should produce a schedule that ensures sufficient bandwidth for each communication while respecting the main property of the all-to-all exchange: no server should send or receive data more than once during a phase. Additionally, it is necessary to optimize the schedule with a minimum number of phases, as detailed in Section 5.4.1. Consequently, to meet these constraints and optimization requirements, we chose to use an Integer Linear Programming (ILP) model.

Usually, the objective of the ILP model is to test all possible solutions to find the most optimized one. In the context of scheduling, an optimization function is not necessary because we aim to create a schedule that avoids congestion in an optimal number of phases but we know the optimal number of phases from the formula proposed in Section 5.4.1. An optimisation function would only slowdown the execution.

As mentioned earlier, we propose an adaptation of the bandwidth-optimal all-to-all schedule [Prisacari *et al.* 2013b] to handle failures. While any scheduling pattern could be adapted to support link failures, we chose the bandwidth-optimal all-to-all pattern because external communications are more evenly distributed between phases compared to the linear-shift. This even distribution of external communications results in fewer communications that need to be reassigned to extra phases, which in turn reduces the number of variables in the ILP model, leading to decreased computation time.

In the ILP model, each communication at each phase is represented by a binary variable $y_{s,d,p}$ that is equal to 1 if there is traffic scheduled between a server source $s \in S$ and a server destination $d \in S$ at phase $p \in P$ and 0 otherwise. S is the set of servers in the fat-tree topology and P is the set of phases. To create the y variables in this model in case of failures, we go through the list of communications to be moved that was computed in Algorithm 3. For example, in Table 5.4, one of the communications S0-S2 and S1-S4 at phase 1 needs to be moved to the new phases $P_{8..11}$. Therefore, in the ILP model, we create the variables $y_{0,2,1}$ and $y_{1,4,1}$, but also the variables $y_{0,2,8..11}$ and $y_{1,4,8..11}$. Then, the model computes all possible solutions where $y_{0,2,1} + y_{0,2,8..11} = 1$ and $y_{1,4,1} + y_{1,4,8..11} = 1$ while respecting the constraints:

1. No source server communicates to the same destination server as another source server during the same phase:
   $\forall d \in S, p \in P, \sum_{s \in S} y_{s,d,p} \leq 1.$

2. At the end of all phases, every server will have communicated with every other

server:
$\forall s, d \in S \sum_{p \in P} y_{s,d,p} = 1.$

3. Each source communicates at most once at each phase:
$\forall s \in S, p \in P, \sum_{d \in S} y_{s,d,p} \leq 1.$

4. The communications from a leaf switch $l \in L$ at phase $p \in P$, $V_{l,p}$, where L is the set of leaf switches, should be less than or equal to the available bandwidth bw_l for the leaf switch $l \in L$.:
$\forall l \in L, p \in P, V_{l,p} \leq bw_l.$

5. The communications to a leaf switch $l \in L$ at phase $p \in P$, $X_{l,p}$, where L is the set of leaf switches, should be less than or equal to the available bandwidth bw_l for the leaf switch $l \in L$.:
$\forall l \in L, p \in P, X_{l,p} \leq bw_l.$

By combining Algorithm 3 with the ILP model, we can adapt the bandwidth-optimal schedule, or any other schedule, to address congestion.

5.5 Results

We evaluate the proposed fault-adaptive scheduling based on the time required to perform an all-to-all exchange, as a shorter all-to-all exchange duration results in higher network throughput. Figures 5.10, 5.11, and 5.12 show the time performance results for the FT(2;16,16), FT(2;20,18), and FT(2;16,32) fat-tree topologies, respectively. The methodology used is the same as presented in Section 5.3. Our solution, Fault-adaptive Scheduling (FS), is compared with the Linear-Shift (LS) and Bandwidth-Optimal (BO) scheduling. The time to complete an all-to-all exchange according to the scheduling algorithm used is computed using Algorithm 2.

Since FS is a scheduling solution that creates no congestion, the only factor that increases the time to complete the all-to-all exchange is the increase in the number of phases. The computation of the number of phases for the FS solution is explained in Section 5.4.1. However, the increase in the number of phases is not significant enough to be worse than leaving congestion unaddressed when the bandwidth reduction is less than or equal to half the available bandwidth.

As shown in Figures 5.10, 5.11, and 5.12, FS demonstrates significantly better performance than LS and BO when the available bandwidth of the fat-tree topology is reduced by up to half. As the bandwidth reduction increases, the number of paths between a given source and destination pair decreases, reducing the possibilities of finding an optimal solution. Consequently, FS approaches the performance of the LS and BO solutions. However, losing more than half of the links in a network is an unrealistic scenario and would be so catastrophic that deploying a fault-adaptive scheduling solution would not be a priority.

In general, FS results in smaller slowdowns than LS and BO for all topologies and scenarios considered. For a bandwidth reduction of 1, the slowdown of FS is approximately 1, meaning that the increase in the number of phases does not significantly impact the time to complete the all-to-all exchange. Although the slowdown increases as the bandwidth reduction increases, it remains significantly lower compared to LS and BO. This makes FS the best of the three solutions.

For the topologies FT(2;16,16), FT(2;20,18), and FT(2;16,32), the slowdown of LS is 1.875, 1.88, and 1.9375, respectively, with a bandwidth reduction of 1. This indicates that it takes almost twice as long to complete an all-to-all exchange with LS compared to no bandwidth reduction. The slowdown of LS continues to grow slightly, approaching 2, as the bandwidth reduction increases.

With a bandwidth reduction of 1, the slowdown of BO is 1, 1, and 1.5 for the topologies FT(2;16,16), FT(2;20,18), and FT(2;16,32), respectively. This result can be explained by the fact that the BO schedule better balances the bandwidth at each phase, which does not necessarily create congestion with a bandwidth reduction of only 1 for the topologies FT(2;16,16) and FT(2;20,18), as shown in Figure 5.5. The topology FT(2;16,32) has fewer available paths, with 32 leaf switches and only 16 spine switches, leading to congestion even with a bandwidth reduction of 1. As the bandwidth reduction increases, the slowdown of BO increases accordingly, reaching approximately 2 for all topologies when only half the bandwidth is available.

The speedup of FS compared to LS is close to 2 with a bandwidth reduction of 1 for all topologies and decreases to 1.02 as the bandwidth reduction approaches half the available bandwidth. Compared to BO, the speedup of FS is 1, 1, and 1.4 for the topologies FT(2;16,16), FT(2;20,18), and FT(2;16,32), respectively, with a bandwidth reduction of 1. It increases to approximately 1.8 for all topologies with a bandwidth reduction of 2 and then decreases to 1.03 as the bandwidth reduction reaches half the available bandwidth.

To conclude, FS demonstrates much better performance in terms of the time to complete the all-to-all exchange compared to LS and BO.

Figure 5.13 shows the computation time required to produce a fault-adaptive scheduling. As explained in Section 5.4.3, to obtain a fault-adaptive scheduling, we adapt the bandwidth-optimal scheduling using Algorithm 3 and an Integer Linear Programming (ILP) model. Algorithm 3 identifies the communications that need to be moved to the additional phases to avoid congestion. Then, the ILP model finds the optimal combination that satisfies bandwidth and the all-to-all exchange constraints in general. The complexity of Algorithm 3 is $O(P \times L)$, where P represents the number of phases and L is the number of leaf switches. This complexity is manageable even for large topologies. For example, with the topology of the LHCb DAQ network FT(2;20,18), $P = 20 \times 18 = 360$ and $L = 18$, resulting in $360 \times 18 = 6480$ iterations, making the computation almost instantaneous. However, the ILP model can be time-consuming.

Figure 5.13 shows the computation time for both Algorithm 3 and the ILP model,

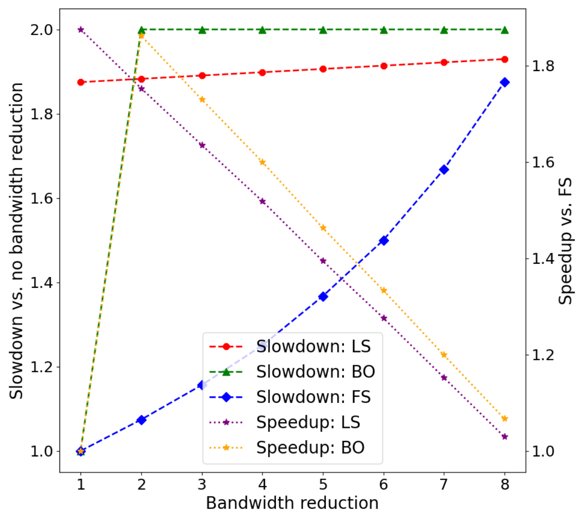


Figure 5.10: Time performance with the topology FT(2;16,16). LS refers to the linear-shift scheduling, BO is the bandwidth-optimal scheduling and FS is our proposed fault-adaptive scheduling. The left axis shows the slowdown of the different scheduling upon bandwidth reduction compared to no failures, this is the relative time degradation due to failures. The speedup, on the right axis, shows the improvement ratio of FS compared to LS and BO.

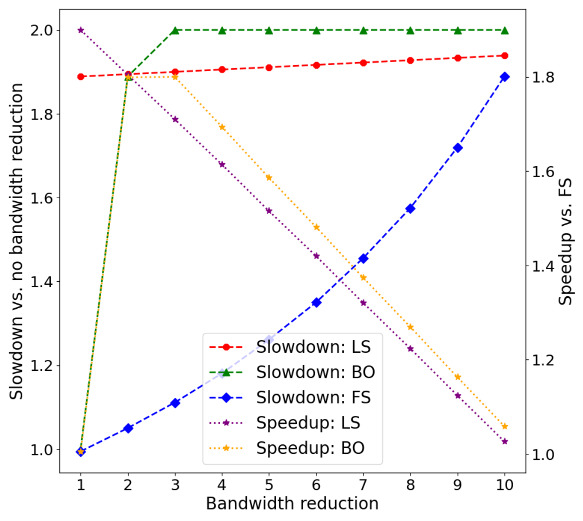


Figure 5.11: Time performance with the topology FT(2;20,18), which is the topology of the LHCb DAQ network. LS refers to the linear-shift scheduling, BO is the bandwidth-optimal scheduling and FS is our proposed fault-adaptive scheduling. The left axis shows the slowdown of the different scheduling upon bandwidth reduction compared to no failures, this is the relative time degradation due to failures. The speedup, on the right axis, shows the improvement ratio of FS compared to LS and BO.

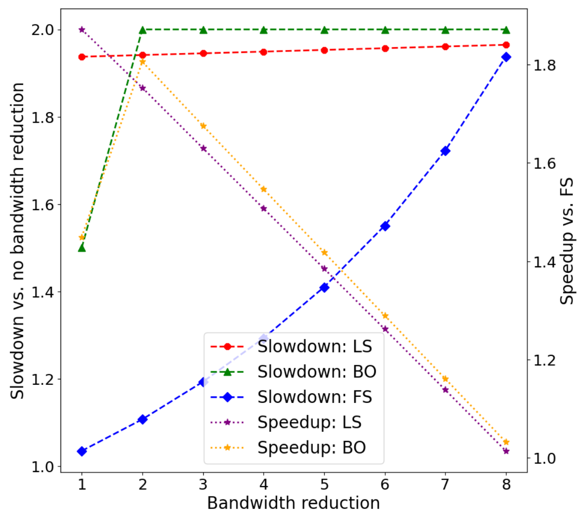


Figure 5.12: Time performance with the topology FT(2;16,32). LS refers to the linear-shift scheduling, BO is the bandwidth-optimal scheduling and FS is our proposed fault-adaptive scheduling. The left axis shows the slowdown of the different scheduling upon bandwidth reduction compared to no failures, this is the relative time degradation due to failures. The speedup, on the right axis, shows the improvement ratio of FS compared to LS and BO.

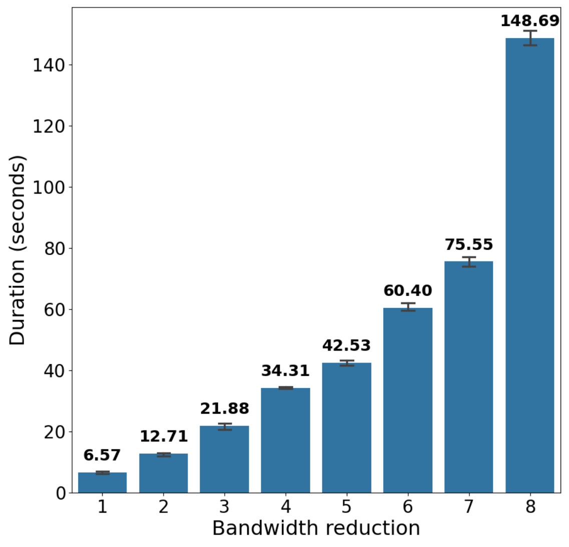


Figure 5.13: The computation time of the fault-adaptive scheduling under bandwidth reduction. The fat-tree topology used is FT(2;20,18), which is the topology of the LHCb DAQ network. For each bandwidth reduction b, tests were repeated 10 times with random failure scenarios involving one to three leaves and b spines. The error bars represent the minimum, mean and maximum values.

illustrating a significant increase up to 148.69 seconds for a bandwidth reduction of 8. Additionally, the number and location of failures are important in this context because the more failures we have, the more communications need to be rescheduled, increasing the complexity of the ILP model. This means that between a failure on a single leaf switch and a failure on all leaf switches, even if the bandwidth reduction is 1 in both scenarios, the computation time will differ since the latter scenario requires rescheduling more communications than the former.

To evaluate the fault-adaptive scheduling in terms of computation time, we randomly simulated 10 failure scenarios for each bandwidth reduction. The error bars in Figure 5.13 represent the minimum, mean, and maximum values. Each failure scenario involved 1 to 3 leaf switches and a number of spines equal to the bandwidth reduction value. We chose this number of failures because the literature shows that groups of failures containing more than 3 simultaneous failures are unlikely, with 80% of groups containing less than 3 simultaneous failures [Gill *et al.* 2011]. We run the ILP model with a "11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz" processor, using 8 threads.

Figure 5.13 shows that the computation time increases significantly with the number of failures. However, fault-adaptive scheduling remains viable for scenarios involving a realistic number of failures.

5.6 Conclusion

In this chapter, we discussed scheduling algorithms for all-to-all traffic matrix. We presented the concept of bandwidth reduction, one of our contributions, which allows us to more precisely define the impact of failures on the routing of an all-to-all schedule. We demonstrate that current scheduling algorithms are inadequate for addressing congestion effectively as the number of phases is not sufficient for the all-to-all exchange under bandwidth reduction. To address failures, we propose increasing the number of phases according to the bandwidth reduction caused by the failure scenario in the network. We propose a formula for computing the lower bound on the number of phases and demonstrate that it increases only slightly until half of the network bandwidth is lost.

With the lower bound on the number of phases, we propose a fault-adaptive scheduling composed of Algorithm 3 and an Integer Linear Programming (ILP) model described in 5.4.3 to adapt any scheduling pattern to failures by identifying the communications that create congestion and rescheduling them in additional phases given by the lower bound on the number of phases. We test this solution by adapting the bandwidth-optimal scheduling for performance reasons, but any scheduling can be adapted to failures.

We then evaluate our fault-adaptive scheduling and compare it with the widely used linear-shift scheduling pattern which is the most optimized scheduling pattern associated with the Infiniband Ftree routing algorithm without failures on the network [Zahavi *et al.* 2009].

Additionally, we compare our solution with the bandwidth-optimal scheduling pattern[Prisacari *et al.* 2013b], which more effectively distributes communications that use the network bandwidth at each phase, showing potential for fault tolerance. Our solution completely avoids congestion, unlike the linear-shift and bandwidth-optimal scheduling patterns, and significantly outperforms them in terms of completion time for the all-to-all exchange.

The primary drawback of our fault-adaptive scheduling solution is its reliance on integer linear programming, which can become time-consuming in terms of computation as the number of network failures increases. For instance, in the studied FT(2;20,18) network topology, finding a scheduling solution with a bandwidth reduction of 8 takes an average of 148.69 seconds. Furthermore, the scenarios considered are relatively simple, and as we will discuss in the next chapter, this approach may not be sufficient to address more challenging failure scenarios. This implies that finding a routing solution using our fault-adaptive scheduling is impossible without causing congestion in these more challenging failure scenarios.

Nevertheless, the proposed solution remains viable for typical failure scenarios and offers the flexibility to adapt any scheduling pattern to better tolerate network failures.

Chapter 6

Fault-adaptive Optimized Routing and Scheduling

Contents

In Chapter 5, we introduced an algorithm associated with an integer linear programming (ILP) model that adapts any all-to-all scheduling solution to address some failure scenarios without causing congestion. However, this approach is not always optimal, as the complexity of the ILP model increases with the number of failures, potentially making it unsolvable for complex failure scenarios.

In this chapter, we identify the failure scenarios that make existing scheduling solutions incapable of avoiding congestion and define the key property that a scheduling algorithm must respect to ensure a congestion-free routing solution. We then 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.

FORS consists of an algorithm to compute the scheduling of the communications for the all-to-all collective exchange in case of link failures. Additionally, FORS includes of a semi-algorithmic routing solution, combining a routing algorithm for

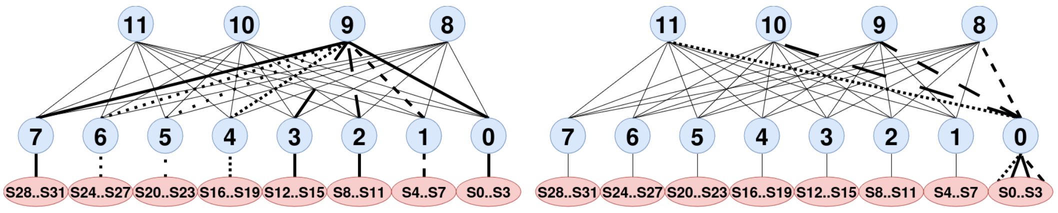


Figure 6.1: Illustration of the deterministic nature of the paths in a two-layer fat-tree topology FT(2;4,8). On the left, all servers have one unique path to join spine 9 represented by different dashed lines. On the right, all spine switches have one unique path to join servers S0..3.

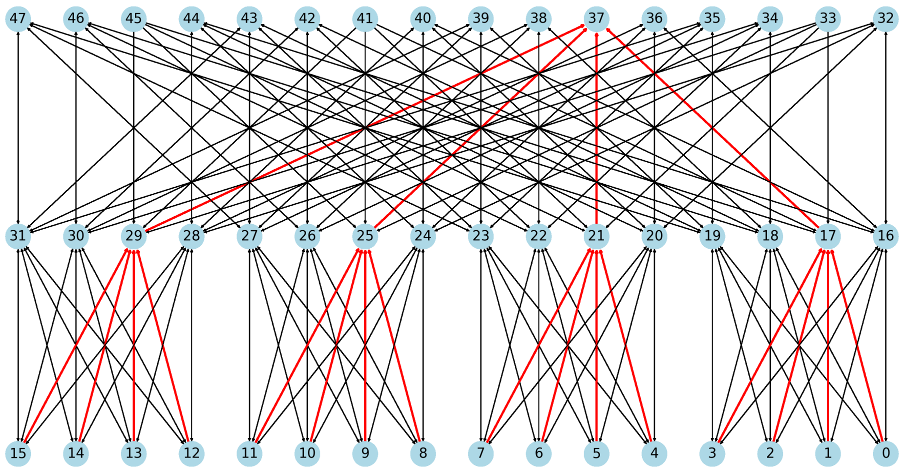


Figure 6.2: Illustration of the deterministic nature of the paths in a three-layer fat-tree topology FT(3;4,4,4). All leaf switches have a single path to join spine 37 (red links), and vice versa.

basic failure scenarios with an Integer Linear Programming (ILP) model with dynamic programming designed to address more challenging combinations of failures. The purpose of the routing algorithm and the ILP model is to provide congestion-free paths between servers in the network based on the given failure scenario. We demonstrate the applicability and performance of our solution on a real, operational data acquisition (DAQ) network that processes large volumes of scientific data. We compare our proposal with currently deployed approaches. Finally, we present the applicability of FORS and the associated deployment constraints.

6.1 Spine pinning problem

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 [Petrini & Vanneschi 1997] fat-tree topology, the shortest path to go from a spine to a destination server is always unique. Figure 6.1 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 $S0 \ldots 3$. This property is valid on any type of k-ary-L-tree fat-tree, even in topologies with more than 2 layers. For instance, Figure 6.2 depicts the fat-tree topology FT(3;4,4,4) with three layers. All leaf switches (0..15) have a single path to join spine 37, as shown by the red links. If we know the spine, the path from the spines to a server remains deterministic, and vice versa, regardless of the scale of the network. Therefore, in a fat-tree topology, the spine switch used to route communication between any source-destination pair of servers defines the entire path.

This property makes the routing problem easy to solve when the bandwidth reduction is uniform across all leaf switches or when there is no reduction in bandwidth. As explained in Section 5.2, a uniform bandwidth reduction in a fat-tree topology occurs when at least one leaf switch experiences $f \geq 1$ failures with the spine switches and the reduction r in the number of available paths between the leaf switch impacted by failures and the other leaf switches is always equal to f, $r == f$. For instance, if the link between switch 0 and switch 8 is broken in Figure 6.1, $f = 1$ and the number of available paths between switch 0 and switches $1 \ldots 7$ is reduced by 1, resulting in $r = 1$ and $r = f$. The concept of non-uniform bandwidth reduction is further discussed in Section 6.2. In this case, the routing problem becomes more challenging to solve in failure scenarios that involve unequal bandwidth reduction between leaf switches, as shown in Section 6.2.

6.2 Non-uniform bandwidth reduction

Figure 6.3 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 via spines 10 and 11, see long dashes in Figure 6.3. 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.

In this scenario, the bandwidth reduction is non-uniform as switch 0 has three paths available to leaf switches 2 through 7, but only two paths to leaf switch 1. The same applies to switch 1. Addressing non-uniform bandwidth reduction is more complex, as we need to consider the specific reduction in paths between certain switches in the all-to-all schedule. This complexity makes the linear-shift [Zahavi *et al.* 2009] and bandwidth-optimal [Prisacari *et al.* 2013b] schedule discussed in Chapter 5 impossible to adapt to such cases. The need to reschedule many communications to

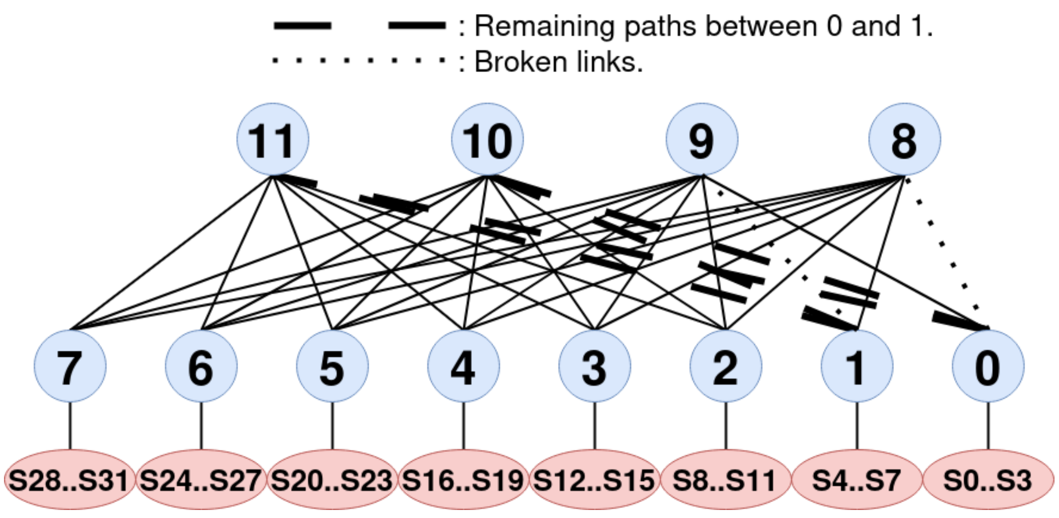


Figure 6.3: Illustration of a non-uniform bandwidth reduction on a two-layer fat-tree topology as it involves an unequal number of paths between different leaf switches. The links 0-8 and 1-9 are broken, resulting in switch 0 having only 2 paths available to join switch 1 and 3 paths available to join the rest of the leaf switches.

additional phases increases computation time significantly, which also completely changes the initial all-to-all schedule.

For instance, consider a phase where the communications between servers 0→4, 1→5, and 2→6 occur in the topology depicted in Figure 6.3. The communications 0→4 and 1→5 can use spines 10 and 11 as paths, respectively. However, there is no available spine for the communication 2→6 because spines 10 and 11 are already being used as paths, and spine 8 is down for leaf 0, while spine 9 is down for leaf 1. As a result, communication 2→6 is forced to share the link between leaf 0 and spine 10, causing congestion. This is an example of a non-uniform bandwidth reduction problem.

The approach of adapting existing schedules to handle such failure scenarios, presented in Section 5.4.3, is insufficient, as their structure fails to resolve the non-uniform bandwidth reduction problem. Algorithm 3 does not fully address this issue, as it requires rescheduling too many communications, leading to increased complexity in the ILP model and, consequently, prolonged computation times. In scenarios where non-uniform bandwidth reduction impacts more than half of the leaf switches, as in the topology shown in Figure 6.3 with broken links 0-8, 1-9, 2-10, 3-11, and 4-8, it becomes unfeasible to find a scheduling solution that enables congestion-free routing using Algorithm 3.

To provide a clearer example, we present the linear-shift scheduling in Tables 6.1 and 6.2 for servers S0 through S3, where the links 0-8, 1-9, 2-10, and 3-11 are broken, in the topology of Figure 6.3. The communications marked with an asterisk must be moved to additional phases, resulting in 28 communications that need to be rescheduled for the leaf switch 0. Similarly, 28 communications must be moved

Table 6.1: The linear-shift scheduling for 32 servers and the first 16 phases, only the communications involving servers S0 through S3 are represented. Communications marked with an asterisk (*) must be moved to additional phases to address the failure scenario with the topology shown in Figure 6.3, where the following links have failed: 0-8, 1-9, 2-10, and 3-11.

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
S0	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15
S1	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16
S2	S2	S3	S4	S5	S6*	S7*	S8	S9	S10*	S11*	S12	S13*	S14*	S15*	S16	S17
S3	S3	S4	S5	S6*	S7*	S8	S9*	S10*	S11*	S12	S13*	S14	S15*	S16	S17*	S18*

Table 6.2: The linear-shift scheduling for 32 servers and last 16 phases, only the communications involving servers S0 through S3 are represented. Communications marked with an asterisk (*) must be moved to additional phases to address the failure scenario with the topology shown in Figure 6.3, where the following links have failed: 0-8, 1-9, 2-10, and 3-11.

	P16	P17	P18	P19	P20	P21	P22	P23	P24	P25	P26	P27	P28	P29	P30	P31
S0	S16	S17	S18	S19	S20	S21	S22	S23	S24	S25	S26	S27	S28	S29	S30	S31
S1	S17	S18	S19	S20	S21	S22	S23	S24	S25	S26	S27	S28	S29	S30	S31	S0
S2	S18	S19	S20	S21	S22	S23	S24	S25	S26	S27	S28	S29	S30	S31	S0	S1
S3	S19*	S20*	S21*	S22*	S23*	S24*	S25*	S26*	S27*	S28*	S29*	S30*	S31	S0	S1	S2

for each of the switches 1, 2, and 3 affected by the failures, given that the linear-shift pattern and that all leaf switches experience the same non-uniform bandwidth reduction. This totals 112 (28×4) communications that need to be rescheduled.

The lower bound on the number of phases given in Section 5.4.1, is $\left\lceil \frac{4\times(4\times8-4}{4-1} \right\rceil =$ 38, which means that we need at least 6 additional phases for the rescheduling of the communications. As discussed in Section 5.4.3, the ILP model creates a communication variable for the phase when the communication is initially scheduled and for each of the additional phases. Therefore, for each communication to be rescheduled, 7 variables are created in the ILP model. This leads to a total of $112 \times 7 = 784$ variables for the rescheduling of the communications, which significantly increases the complexity of the problem, even for a small topology like FT(2;4,8). If the problem becomes more constrained, such as in scenarios where non-uniform bandwidth reduction impacts more than half of the leaf switches, finding a feasible solution, meaning a solution that enables a routing to be found without creating congestion, becomes impossible without completely changing the initial schedule.

Therefore, a better approach is to develop a new scheduling algorithm that more effectively considers the routing constraints. A key property of this new fault-tolerant scheduling algorithm, designed to handle all types of failure scenarios, is that it limits communications between leaf switches to a single exchange per phase.

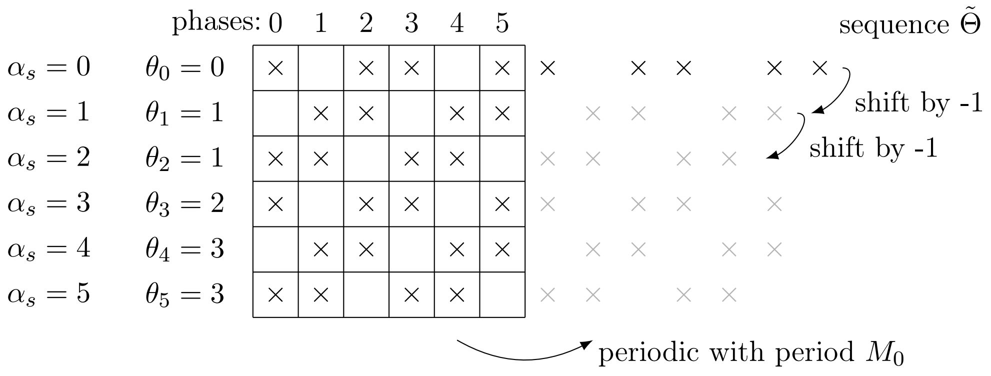

phases: 0 1 2 3 4 5
sequence $\tilde{\Theta}$
$\alpha_s = 0$ $\theta_0 = 0$
$\alpha_s = 1$ $\theta_1 = 1$
$\alpha_s = 2$ $\theta_2 = 1$
$\alpha_s = 3$ $\theta_3 = 2$
$\alpha_s = 4$ $\theta_4 = 3$
$\alpha_s = 5$ $\theta_5 = 3$
shift by -1
shift by -1
periodic with period M_0

Figure 6.4: Illustration of function $T(s,p)$, with $M_0 = 6$ and $f = 2$, that defines whether a server s (the lines) transmits at phase p or not. The function depends only on α_s and is periodic of period M_0. Server s transmits at phase p if and only if the corresponding cell is crossed.

This means that at each phase, each leaf switch communicates at most once with another leaf switch. This property ensures that a routing solution can always be found, regardless of the bandwidth reduction between any two leaf switches, as long as they remain connected in the topology.

6.3 Fault-adaptive scheduling algorithm

The work presented in this section was conducted in collaboration with Quentin Bramas, an Associate Professor at the University of Strasbourg, France. He designed Algorithm 5 and demonstrated its functionality, as detailed in this section. The implementation of the algorithm and the properties it needs to respect were carried out by Éloïse Stein, the author of this thesis.

In this section, we present our new scheduling algorithm performing a congestion-free all-to-all exchange in an arbitrary two-layer fat-tree. The challenge is to use the 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 k-ary-L-tree the root is replaced by M_0 spine switches, each connected to all the leaf switch and links having equal bandwidth [Pippenger 1978]. In this case, f is the bandwidth reduction, *i.e.*, the maximum number of failed links on a leaf switch. The bandwidth reduction f is computed by taking the number of link failures over all leaf switches and finding the maximum of these numbers. It is known [Prisacari *et al.* 2013b] 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 [Prisacari *et al.* 2013b] works without being impacted by faults as illustrated in Figure 5.5.

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 as shown in Chapter 5.

The index of a server $s \in [0, M_0M_1]$ (line 15 in Algorithm 5) 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 (line 13 in Algorithm 5) and $\alpha_s \in [0, M_0 - 1]$ is the index of the server among the servers below the leaf switch. For instance, in the topology $FT(2; 4, 8)$ with $M_0 = 4$, $M_1 = 8$ depicted in Figure 6.3, the index s of server 22 can be computed as follows:

$$s = \alpha_s + \gamma_s M_0 = 2 + 5 * 4 = 22$$

.

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 $i \in [0 \ldots P - 1]$ is the index in the sequence. The computation of $\tilde{\Theta}$ is given in lines 7-9 in Algorithm 4. For instance, in the topology $FT(2; 4, 8)$ with $M_0 = 4$, $M_1 = 8$, $f = 1$ and the number of servers $P = M_0 \times M_1 = 4 \times 8 = 32$, the computation of $\tilde{\Theta}(i)$ with $i \in [0, 1, 2, .., P - 1]$ is :

$$i = 0, \tilde{\Theta}(0) = \left\lceil \frac{0 \times 4}{4 - 1} \right\rceil = 0$$

$$i = 1, \tilde{\Theta}(1) = \left\lceil \frac{1 \times 4}{4 - 1} \right\rceil = 2$$

$$i = 2, \tilde{\Theta}(2) = \left\lceil \frac{2 \times 4}{4 - 1} \right\rceil = 3$$

$$\ldots$$

$$i = P - 1 = 31, \tilde{\Theta}(31) = \left\lceil \frac{31 \times 4}{4 - 1} \right\rceil = 42$$

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 to ensure that the first phase is non-negative). The computation of Θ is given at lines 13-17 in Algorithm 4 and the computation of θ in lines 10-12. For instance, in the topology $FT(2; 4, 8)$ with $M_0 = 4$, $M_1 = 8$, $f = 1$, the number of servers $P = M_0 \times M_1 = 4 \times 8 = 32$ and the servers with index 2 in the leaf switches to which they are connected, $\alpha_s = 2$, θ_s is $\left\lfloor \frac{(2-1)(4-1)}{4} \right\rfloor + 1 = 1$. In this case, $\theta = [0, 1, 1, 2]$ for all possible index of the servers in the leaf switches they are connected to which are $[0, 1, 2, 3]$. In the example with $\alpha_s = 2$, the set of phases when the servers can transmit with an index of two in the leaf switches they are connected to is Θ_i with $i \in [0..P - M_0 - 1]$ and is computed as follows:

$$\begin{aligned} i = 0, \Theta_i &= \tilde{\Theta}(i + \theta_{\alpha_s}) - \alpha_s = \tilde{\Theta}(0 + 1) - 2 = 2 - 2 = 0 \\ i = 1, \Theta_i &= \tilde{\Theta}(i + \theta_{\alpha_s}) - \alpha_s = \tilde{\Theta}(1 + 1) - 2 = 3 - 2 = 1 \\ &\ldots \\ i = 27, \Theta_i &= \tilde{\Theta}(i + \theta_{\alpha_s}) - \alpha_s = \tilde{\Theta}(27 + 1) - 2 = 38 - 2 = 36 \end{aligned}$$

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 6.4 presents an example of a function T in a fat-tree $FT(2; 6, 6)$ with a bandwidth reduction of $f = 2$.

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 $T(s, p) = 1$ (line 16 in Algorithm 5), then the destination server is computed using the number $N_T(s, p) + \theta_s$ (the lines 19-25 in Algorithm 5). This number is used to compute the destination leaf D_{leaf} (among the $M_1 - 1$ possible destination leaves, line 21 in Algorithm 5) and the index of the destination server below this leaf D_{server} (among the M_0 possible servers, line 22 in Algorithm 5). The destination index $D_{leaf} M_0 + D_{server}$ (line 23 in Algorithm 5), 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:

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

The scheduling algorithm is described in Algorithm 4 and Algorithm 5. 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 $P = M_0M_1$ the number of servers.

Algorithm 4: The computation of θ and Θ, which are necessary for Algorithm 5. 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.

```
1  Input: The number of servers $M_0$ connected to a leaf switch.
2  Input: The number of leaf switches $M_1$
3  Input: The bandwidth reduction $f$.
4  Output: The sequence $\theta$ of size $M_0$
5  Output: The sequence $\Theta$ of size $M_0$
   // Computation of the number of servers $P$
6  $P = M_0 \times M_1$
   // Computation of the sequence $\tilde{\Theta}$ of evenly distributed integers
7  $\tilde{\Theta}$ := Empty list of size $P$
8  foreach $i \in [0, \dots, P-1]$ do
9      $\tilde{\Theta}[i] = \left\lceil \frac{iM_0}{M_0-f} \right\rceil$
   // Computation of the starting indexes $\theta$ from the sequence $\tilde{\Theta}$ for all
      possible index of the servers in the leaf switches they are connected to.
10 $\theta$ := Empty list of size $M_0$
11 foreach $\alpha_s \in [0, \dots, M_0-1]$ do
12     $\theta[\alpha_s] = \left\lfloor \frac{(\alpha_s-1)(M_0-f)}{M_0} \right\rfloor + 1$
   // Computation for each server of the set of phases $\Theta$ when it transmits.
13 $\Theta$ := Empty list of size $M_0$
14 foreach $\alpha_s \in [0, \dots, M_0-1]$ do
15     $\Theta[\alpha_s]$ := Empty list of size $P - M_0$
16     foreach $i \in [0, \dots, P-M_0-1]$ do
17         $\Theta[\alpha_s][i] = \tilde{\Theta}[i + \theta[\alpha_s]] - \alpha_s$
18 return $\theta$, $\Theta$
```

Algorithm 5: 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. θ and Θ are computed in Algorithm 4. The output is *schedule* which is the computed scheduling that gives the ordered list of source-destination pairs for each phase.

```
1  Input: The number of servers $M_0$ connected to a leaf switch.
2  Input: The number of leaf switches $M_1$
3  Input: The bandwidth reduction $f$
4  Output: The dictionary $schedule$ with the phase $p$ as key and the list of
     tuple $(s, d)$ as value, where $s$ is the source server and $d$ is the destination
     server
   // Computation of the number of servers $P$
5  $P = M_0 \times M_1$
   // Computation of the number of phases upon bandwidth reduction
6  $nbPhase = \left\lceil \frac{(P-M_0)*M_0}{M_0-f} \right\rceil$
   // Number of transmissions before returning to the same destination. The
      lcm function returns the least common multiple between $M_1 - 1$ and $M_0$
7  $cycleLength = lcm(M_1 - 1, M_0)$
   // Number of times $N_T^s$ a server $s$ has transmitted data
8  $N_T$ := List of size $P$ filled with the value 0
9  $schedule$ := Empty list of size $nbPhase$
10 foreach $p \in [0, \ldots, nbPhase - 1]$ do
11     $scheduling[p]$ := Empty list of size $P$
12     foreach $sourceLeaf \in [0, \ldots, M_1 - 1]$ do
13         $\gamma = sourceLeaf * M_0$
14         foreach $\alpha \in [0, \ldots, M_0 - 1]$ do
               // Computation of the source server $s$
15             $s = \gamma + \alpha$
16             if $p \in \Theta[\alpha]$ then
17                 if $N_T[s] \geq P - M_0$ then
                       // The server has finished transmitting
18                     continue
19                 $n = N_T[s] + \theta[\alpha]$
20                 $cycleOffset = \left\lfloor \frac{N_T[s]}{cycleLength} \right\rfloor$
21                 $groupDest = n \mod (M_1 - 1)$
22                 $offsetDest = (n + cycleOffset) \mod M_0$
23                 $shift = groupDest * M_0 + offsetDest$
24                 $N_T[s] = N_T[s] + 1$
25                 $d = (\gamma + M_0 + shift) \mod P$
26                 $scheduling[p].insert((s, d))$
27 return $schedule$
```

6.4 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 onto paths. 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 6.

6.4.1 Routing algorithm

The objective of our routing algorithm is to efficiently assign a unique spine to each source-destination pair at 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 [Petrini & Vanneschi 1997]. To prevent multiple communication flows from using 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.

We propose Algorithm 6 to compute a unique spine to each source-destination pair at 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 6 is simply to exclude spines that contain at least one failure from being used as paths for source-destination pairs. This property makes Algorithm 6 only applicable when the number of spines impacted by failures m is inferior or equal to the bandwidth reduction f $(m \leq f)$, *i.e.*, the bandwidth reduction is uniform. For instance, in Figure 6.3 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 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 6 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 6.3. The selected spine is identified by the index i in Algorithm 6.

Consider a generalized fat-tree with $n = M_0$ spines and consider the array of n spines denoted $N = (N_0, \dots, N_{n-1})$. Let $p \in P$ be a phase, P be the ordered list of phases and $D_l^p = (d_0^l, d_1^l, \dots, d_{k-1}^l)$ 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 $S_{i \bmod m} \in S$ 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 and communicating in the same phase 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 servers that communicate 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 6.3 are shifted by one.

The solution computed by Algorithm 6 can be found in polynomial time. The complexity of the routing algorithm is the same as the complexity of the scheduling algorithm. The algorithm walks through all source-destination pairs to assign each a spine, resulting in a complexity of $O(N^2)$, where N is the number of servers in the topology.

However, in the case of other failure scenarios, in particular, if m spines are impacted by failures, with $m > f$ where f is the bandwidth reduction, as illustrated in Figure 6.3, 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).

6.5 Integer Linear Programming with dynamic programming

In addressing failure scenarios with a non-uniform bandwidth reduction, *i.e.*, where the number of spines impacted by failures is strictly superior to the bandwidth reduction f, the formulation of a general routing algorithm appears, to our best understanding, unattainable. In response, our approach is to use an ILP model with dynamic programming to find a feasible routing solution, *i.e.*, without conflicts on the links in the network.

We believe that a general routing algorithm can not be found, as our problem is similar to the Multi-Commodity Flow problem [Karp 1975]. The multi-commodity flow problem involves routing multiple flows (or commodities) through a network from their respective sources to their destinations, while respecting capacity constraints on the links [Karp 1975]. The objective of a multi-commodity flow problem is typically to minimize or maximize the costs associated with the links while satisfying these constraints. This is similar to our problem, as we want to minimize the cost of routing each communication between a source server and a destination

Algorithm 6: 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_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 5 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).

```
1  Input: The dictionary $schedule$ with the phase $p$ as key and the ordered
     list of source-destination pairs $(s, d)$ as value.
2  Input: The number of servers $M_0$ connected to a leaf switch
3  Input: The ordered list $S$ of spine switches that do not contain any failed
     links
4  Output: The dictionary $routingTable$ with the destination pair $(s, d)$ as
     key and the spine to use in the path to go from $s$ to $d$ as a value
5  $routingTable$ := Empty dictionary of size $N^2$, where $N$ is the number of
     servers in the topology
6  foreach $p \in schedule$ do
       // $i$ is the index of the spine in S.
7      $i = 0$
8      foreach $(s, d) \in schedule[p]$ do
           // Computation of the source leaf switch $sLeaf$ and the destination
              leaf switch $dLeaf$ with the source server $s$ and the destination
              server $d$
9          $sLeaf = \left\lfloor \frac{s}{M_0} \right\rfloor$
10         $dLeaf = \left\lfloor \frac{d}{M_0} \right\rfloor$
           // If the communication $(s, d)$ is not local to a leaf switch.
11         if $sLeaf \neq dLeaf$ then
               // $S[i]$ is the $i$th spine in the ordered list of spines $S$ that do
                  not contain any failed links.  It is assigned to communication
                  $(s, d)$ to be used in the path from source $s$ to destination $d$.
12             $routingTable[(s, d)] = S[i]$
13             $i = (i + 1) \mod |S|$
14 return $routingTable$
```

server.

More specifically, this means we want to ensure that a communication flow does not go through the top layer of the fat-tree topology more than once to join a destination. Additionally, our routing must respect certain constraints, which we describe in Section 6.5.1.2, to avoid congestion, meaning that a link should not be used more than once in the same direction.

The multi-commodity flow problem is known to be NP-complete [Karp 1975] and, in some variants, NP-hard. A variant of the multi-commodity flow problem is the Unsplittable Flow Problem (UFP) [Kleinberg 1996], in which each flow must be routed along a single path from its source to its destination. This means the flow can not be split across multiple paths. This problem is NP-hard and could relate to our routing problem, as a communication flow between a source and a destination can not be split into multiple paths and must use a single path. Since our problem could be NP-complete or NP-hard (such a proof is left for future work), and we could not find an algorithm, we decided to use integer linear programming to solve it.

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 [Prisacari *et al.* 2013c, Schweissguth *et al.* 2017] 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 larger topologies.

6.5.1 The ILP model

Our routing problem involves finding a path in a fat-tree topology for each source-destination pair to provide a routing solution without congestion. In general, the constraints that our routing must respect are:

- Unique spine assignment: For each source-destination pair, exactly one spine must be selected.

- Avoiding upstream congestion: For each source leaf, no more than one communication can use the same spine.

- Avoiding downstream congestion: For each destination leaf, no more than one communication can use the same spine.

All the variables in the model needs to be binary and linear constraints are necessary to express the problem. Our constraints are linear because they involve summing binary variables and setting these sums to specific values. For instance, the constraint 1 requires the sum of binary variables to equal 1.

ILP is characterized by decision variables that are restricted to integer values, such as binary values, and by constraints that must be linear. Our problem aligns

Table 6.3: Notations in the ILP model.

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

perfectly with these characteristics, making it well-suited for solving using ILP solvers like Gurobi.

The definition of the problem and structure can be easily expressed with linear formulations. This makes, to our knowledge, Integer Linear Programming (ILP) the most suitable approach. However, ILP problems are known to be NP-complete and challenging to solve, particularly when all variables are binary [Karp 1972]. To address this, we reduced the problem as a spine pinning problem as explained in Section 6.1 and used dynamic programming as shown in Section 6.5.2 to solve it with, to our knowledge, the most efficient solver available: Gurobi. Gurobi is the most optimal solver for ILP problems and offers free licenses for academic use [experts 2023a]. Gurobi demonstrates significantly shorter computation times compared to other widely used solvers such as IBM CPLEX and lpSolve [Luppold *et al.* 2018].

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 6.3.

6.5.1.1 The variables

The variables in our model indicate which spine is selected for each source-destination pair. They are defined by $v(s, d, sp)$, a binary variable where $s \in S$ is the source server, $d \in S$ is the destination server, and $i \in I$ is a spine. This variable takes the value 1 if spine i is chosen for the path between source server s and destination server d, and 0 otherwise. For instance, if during a phase, source server 0 uses spine 8 to reach destination server 10, the variable $v(0, 10, 8)$ is set to 1 in the ILP model, while for all other spines $i \neq 8$, $v(0, 10, i)$ is 0.

6.5.1.2 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:
  $\forall s, d \in S, \sum_{i \in I} v(s, d, i) = 1.$

- At most one spine is used from each source leaf switch to avoid upstream congestion:
  $\forall l \in L, i \in I, \sum_{s \in S_l, d \in S} v(s, d, i) \leq 1.$

- At most one spine is used to each destination leaf switch to avoid downstream congestion:
  $\forall l \in L, i \in I, \sum_{s \in S, d \in S_l} v(s, d, i) \leq 1.$

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

To minimize the computation time of the ILP model, our next objective is to apply dynamic programming which is an optimization technique that divides the problem into smaller sub-problems, each of which is solved using the solutions found for the previous sub-problems.

6.5.2 Dynamic programming

Dynamic programming keeps the computation time reasonable and enables finding solutions for even larger topologies. The idea of dynamic programming is to decompose the problem into smaller sub-problems and use the solutions of these sub-problems to solve the subsequent ones. As described by Prisacari et al. [Prisacari *et al.* 2013c], an Integer Linear Programming (ILP) model is proposed to find optimized paths between a source and a destination within a fat-tree topology, leveraging dynamic programming techniques. The objective in this model is to distribute the load as evenly as possible across the various links at each layer in the fat-tree topology. The approach involves splitting the routing problem by layers within the fat-tree and solving the problem for each layer. At each step, the solution for the current layer uses the routing solution of the previous layer. After solving the problem for each layer, the overall solution is further optimized to ensure that the load on the links is balanced. The approach of Prisacari et al. [Prisacari *et al.* 2013c] aim to find the most optimized solution in terms of load balancing while our objective is feasibility in solving our routing problem. Since our scheduling solution is already optimized to prevent network congestion, using a solution that balances the load across links is not interesting for our problem.

To apply dynamic programming, our problem is divided into distinct phases. At each phase, an ILP model is formulated to compute the routing solution for that specific phase. The solution obtained from the ILP model in the current phase is saved and subsequently used by the ILP models in the following phases.

For each solved ILP model, we save the entries, an entry is defined by the spine assigned to route the communication from one leaf switch to another, as specified in the routing table of the leaf switch. For every source-destination pair (s, d) in the solved ILP model, we compute the source leaf ls (the leaf switch connected to s) and the destination leaf ld (the leaf switch connected to d). These entries are stored in a dictionary with each (ls, ld) as key and the assigned spine as value.

In the current ILP model to be solved, we iterate through each source-destination pair (s, d) and each possible spine $i \in I$ that can be used to route (s, d) (refer to Table 6.3 for notation). We then create the corresponding variable $v(s, d, i)$ in the ILP model. Once the variable is defined, we check if spine i has been previously assigned for routing between the source leaf connected to s and the destination leaf connected to d in the previous phases. If it has, we use the **Start** parameter available in Gurobi [experts 2023c]. The Start parameter allows us to provide an initial feasible solution to the solver, which accelerates the optimization process, especially for complex problems. Consequently, we set the "Start" parameter for variable $v(s, d, i)$ to 1.

The dynamic programming approach has been chosen due to its effectiveness in optimizing computation time, as demonstrated in [Prisacari *et al.* 2013c].

6.6 Results

In this section, we evaluate the throughput achieved by our Fault-adaptive Optimized Routing and Scheduling (FORS) solution, our combined scheduling and routing solutions, on the use-case operational network. We deployed a prototype of our solution and conducted the measurements on the live network. We compare FORS with the state of the art solutions to prove that FORS degrades more gracefully in the event of bandwidth reduction than the other solutions. Finally, we present the computation time required to find a routing solution in complex failure scenarios, necessitating solving our ILP model.

6.6.1 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 leaf switches depicted in Figure 2.7 are noted $SW_{0..17}$ and the spine switches are noted $SW_{18..37}$. 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 2.7:

- 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, *i.e.*, a uniform bandwidth reduction.
- xF $SW_{0,5,11}$: $x \in [1, 2, 3]$ failure(s) between the leaf switches SW_0, SW_5 and SW_{11} and random 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 than a single leaf inside the network and the impact of a non-uniform bandwidth reduction.
- $SW_{18,19}$: 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[Gill *et al.* 2011, Yigitbasi *et al.* 2010, Singh *et al.* 2021] failure scenarios. The literature [Gill *et al.* 2011] shows that grouped failures rarely consist of more than five failures simultaneously. Our failure scenarios range from 1 failure (1F 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.

We measure the impact of bandwidth reduction for different configurations of the DAQ network. The concept of bandwidth reduction is described in Section 5.2. As explained in Section 2.2.3, 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 [Hensgen *et al.* 1988]. This approach has demonstrated better performance than non-synchronized communications without failures, especially in large-scale systems [Pisani *et al.* 2023a]. However, as demonstrated in Section 4.3.4, synchronization impacts throughput in the presence of failures and can degrade performance more than the absence of synchronization. Therefore, we also evaluate the approach without synchronization. The non-synchronized all-to-all application is the same as presented in Section 4.3.1. In this naive approach, each destination requests data from a configurable amount of sources at the same time, letting the network handle the resulting congestion.

The routing algorithm used in the absence of failures is Ftree, as explained in Section 2.3.3, which is the optimal routing algorithm for the linear-shift scheduling [Zahavi *et al.* 2009]. The routing algorithm Ftree is used 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 pure fat-tree topology, and which is recommended as it balances the traffic better than Ftree in case of bandwidth reduction [Drung & Rosenstock 2017] as we show in Section 4.2. 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 [Nvidia 2023b]. Bogdanski et al. [Bogdanski *et al.* 2012] study the performance of these algorithms and show that they are all highly sensitive to network failures and we also demonstrate the performance of Up-Down in Section 4.2.

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 include the adaptive routing version of Ftree (Ftree-AR) into our measurements. This adaptive routing version of Ftree is already implemented in Infiniband [Nvidia 2023b] 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. We compare these routing approaches to FORS.

6.6.2 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 6.6.1.

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 6.3 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 [Nvidia 2023b], 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 server in the routing tables of the switches. This means that instead of having one LID per server, we also encode the spine to be used to reach the server in the LID. Thus, each switch has 20 LIDs to join each server, with every spine in the network having an assigned ID encoded

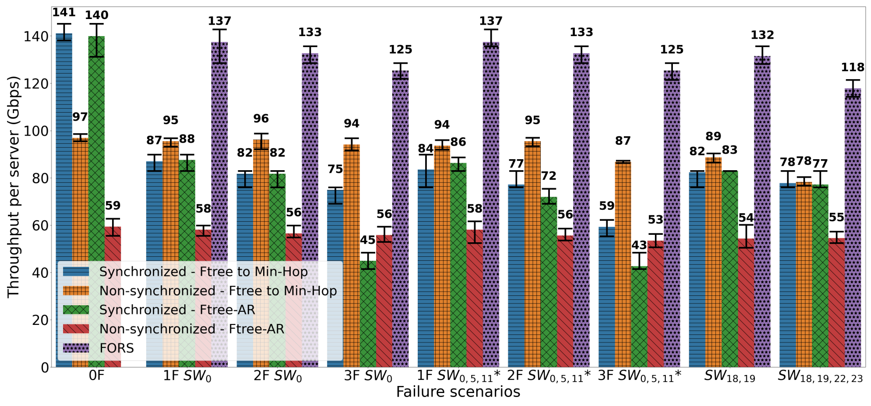


Figure 6.5: Evaluation of FORS throughput compared to the default routing algorithms Ftree/Min-Hop and the adaptive routing version of Ftree with synchronized and non-synchronized all-to-all exchange. To obtain the total throughput achieved by the DAQ application, one can multiply the throughput per 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 for synchronized all-to-all and five minutes for non-synchronized all-to-all. The fat-tree topology of the studied data acquisition network is FT(2;20,18), illustrated in Figure 2.7. The failure scenarios noted with "*" are the ones where an ILP model was needed to compute the routing solution.

within the LID. Using multiple LIDs for routing has already been implemented and has proven to be effective [Lin *et al.* 2004].

The parameter to set up multiple paths to join a destination server in InfiniBand is the LID Mask Control (LMC) [Nvidia 2023b]. The LMC value needed to implement the routing solution is 5, allowing at most $2^5 = 32$ routes for each destination server, which works well with a topology of our size that has 20 spines. 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 routing assigns the LID to use for each communication at each phase, thus defining the complete route for sending traffic. The scheduling of FORS is implemented in the MPI application deployed in the network.

The throughput achieved by FORS decreases more gracefully than the other solutions and depends solely on the bandwidth reduction. The failure scenarios $1F\ SW_0$ and $1F\ SW_{0,5,11}$ exhibit the same throughput despite having a different number of failures which are one and three, respectively. This similarity is due to the throughput achieved by FORS depends solely on the bandwidth reduction, which is one of our contribution described in Section 5.2. FORS computes a congestion-free solution that requires the same number of phases in scenarios $1F\ SW_0$ and $1F\ SW_{0,5,11}$ as the bandwidth reduction for both scenarios is one. Consequently, the unique parameter that fluctuates the throughput is the number of phases which depends on the bandwidth reduction as explained in Section 5.4.1. The number of phases increases with bandwidth reduction to prevent congestion; therefore, a higher number of phases results in a longer all-to-all completion time. This is true only for FORS. For the other solutions, the number of phases is unchanged, and throughput is affected by conflicts causing congestion. The same applies to scenarios $2F\ SW_0$, $2F\ SW_{0,5,11}$ and $SW_{18,19}$, where the bandwidth reduction is 2 even though the actual number of failures differs (2, 6 and 40, respectively). This characteristic allows FORS to maintain the same throughput, whether dealing with the loss of two spine switches (scenario $SW_{18,19}$) or the loss of two links on a single leaf switch (scenario $2F\ SW_0$).

Without bandwidth reduction, the synchronized all-to-all is better than the non-synchronized all-to-all. As previously demonstrated in Section 4.3.3, the synchronized all-to-all shows much better performance without bandwidth reduction in the network (scenario 0F). This performance difference is due to the synchronization preventing congestion. Since each source waits for all sources to finish the current phase before sending to the next destination, the linear-shift scheduling is always respected. In contrast, the non-synchronized all-to-all performs poorly without bandwidth reduction with Ftree and Ftree-AR compared to the synchronized all-to-all. This is due to the accumulated congestion during the all-to-all exchange, resulting in a degradation of the throughput.

With bandwidth reduction, the non-synchronized all-to-all is better than the synchronized all-to-all. As previously highlighted in Section 4.3.4, when congestion becomes unavoidable, i.e. in the case of bandwidth reduction, each synchronized phase has to wait for the congestion of the previous phase to be resolved. In contrast, non-synchronized communications are performed as soon as possible, enabling more effective utilization of the remaining bandwidth in the network.

In the synchronized all-to-all exchange, the bandwidth reduction is uniform between all servers. The synchronized version shows the same throughput with two failures on a single leaf switch and with the loss of two spines. Notably, the throughput with the failure of two spines (scenario $SW_{18,19}$ in Figure 6.5) is better than the throughput with three failures on a single leaf switch (scenario 3F SW_0 in Figure 6.5). This is due to the exchange being synchronized, resulting in a uniform bandwidth reduction for all servers. Whether it is the loss of two links on one leaf switch or the loss of two links on all leaf switches, all servers experience an equivalent bandwidth reduction.

The bandwidth reduction is given by the failure scenarios and two failure scenarios with the same bandwidth reduction experiment the same throughput in the synchronized all-to-all exchange. The failure scenarios 2F SW_0 and $SW_{18,19}$ shows the same throughput while 2F SW_0 is two failures on the leaf switch SW_0 and $SW_{18,19}$ is 40 failures because this is the failure of two spine switches. This is due to the concept of bandwidth reduction described in Section 5.2. Although there is a difference of 38 failures between scenarios 2F SW_0 and $SW_{18,19}$, both scenarios experience the same bandwidth reduction which is 2. As a result, the achieved throughput is almost identical. It is also interesting to note that scenarios where failures are not located on the same leaf switch exhibit lower throughput. For instance, scenario 1F SW_0, representing a failure on a single leaf switch, demonstrates higher throughput than scenario 1F $SW_{0,5,11}$, where three leaf switches experience one link failure with different spines. This behavior becomes more pronounced as the number of failures increases. This is due to the distinction between a uniform bandwidth reduction defined in Section 6.1 and non-uniform bandwidth reduction presented in Section 6.2. The difference between the throughput of the scenarios 1F $SW_{0,5,11}$ and 1F SW_0 arises from the fact that in the scenarios $X \in [1, 2, 3]$F $SW_{0,5,11}$, the bandwidth reduction in non-uniform. While in scenarios $X \in [1, 2, 3]$F SW_0, the bandwidth reduction is uniform. In failure scenarios $X \in [1, 2, 3]$F $SW_{0,5,11}$ with a non-uniform bandwidth reduction, the number of paths between the leaf switches impacted by failures is further reduced as explained in Section 6.2, which leads to reduced throughput compared to failure scenarios $X \in [1, 2, 3]$F SW_0 with a uniform bandwidth reduction.

The throughput of the non-synchronized all-to-all degrades slowly. The non-synchronized all-to-all shows nearly the same throughput in all failure scenarios. The performance of the non-synchronized all-to-all communication starts to degrade with an increasing number of failures, as seen in scenarios such as $3FSW_{0,5,11}$ (loss of three links on three leaf switches), as well as $SW_{18,19}$ and $SW_{18,19,22,23}$ (loss of spine switches).

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. 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. In the case of the non-synchronized all-to-all, Ftree and Min-Hop always perform better than Ftree adaptive routing, which performs poorly. This is attributed to the fact that the traffic generated by the DAQ application is too bursty as explained in Section 4.2.2.

FORS outperforms all routing and scheduling solutions, significantly improving the throughput compared to these existing approach. For instance, FORS increases the throughput per server from 87-88 Gbps with the current approach to 137 Gbps with the failure scenario 1F SW_0. With 326 servers in the DAQ network, this improvement results in a total network throughput increase from 28 Tbps to 44 Tbps on average. As demonstrated in Chapter 4, flapping links exist in the network and often lead to prolonged failures that could last for several hours. FORS would be an interesting approach to address flapping links, as the moment a link starts flapping, we could simply deactivate it temporarily and use FORS until the link becomes stable again. This could result in significantly less data loss compared to leaving the link unstable.

6.6.3 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, in scenarios with a uniform bandwidth reduction, a routing solution can be found in polynomial time, as they are computed by the routing algorithm described in 6.4.1. Furthermore, one potential approach is to pre-calculate a routing solution for all complex failure scenarios.

6.7 Deployment

In this section, we discuss the applicability of FORS to InfiniBand networks. Specifically, we address the number of entries required in the routing tables of switches to reach a destination in the routing solution produced by FORS. It is necessary for the routing solution computed by FORS to utilize multiple paths between a switch and a destination server to avoid congestion. However, the current solution is not optimized in terms of the number of entries, even though it remains applicable to the studied data acquisition network. We propose a Mixed-Integer Linear Programming (MILP) model to compute a routing solution that minimizes the number of entries in the routing tables of the switches. Finally, we discuss the limitations of InfiniBand for deploying FORS on the studied network.

6.7.1 Operational constraints

The limited flexibility of Infiniband imposes some constraint on the current implementation of FORS. 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 [Nvidia 2023b]. 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. However, this limitation does not exist in Ethernet networks; therefore, FORS is readily deployable on these networks. In the LHCb DAQ network, Infiniband was chosen over Ethernet primarily due to Ethernet's lack of flow control mechanisms and the unavailability of 200 Gbps Ethernet network cards at the time the network was designed [Krawczyk, Rafał Dominik *et al.* 2021].

6.7.2 LID Mask Control for Infiniband Multipath

The routing strategy used by FORS needs multiple routes to the same destination server from a leaf switch. In the DAQ network topology FT(2;20,18) shown in Figure 2.7, each leaf switch S is directly connected to twenty servers. Each of these servers must communicate with a destination server d at different phases in the schedule, as simultaneous communications would create congestion. Consequently, the spine used as a path to join server d may vary across different phases, resulting in multiple routes being assigned from the leaf switch S to join server d.

In an InfiniBand network, each port is assigned a unique Local Identifier (LID), which the controller uses to route the packets within the network[Nvidia 2023b]. A LID is a 16-bit value, allowing for a range of 0 to 65,535 possible addresses. The LID Mask Control (LMC) value enables the assignment of multiple LIDs to a same destination, allowing multipath. The LMC value specifies how many of the least significant bits of the 16-bit Destination LID field in the Infiniband packet header are masked [Nvidia 2023a].

For example, with a LMC value of 2, the mask would be `1111111111111100`, or `0xFFFC` in hexadecimal. The two least significant bits are masked, resulting in $2^2 = 4$ distinct addresses for a given destination LID and thus 4 different paths. If the destination LID value is 4, the four possible LIDs would be:

- `0000000000000100`
- `0000000000000101`

- `0000000000000110`
- `0000000000000111`

The LMC value can range from 0 to 7[et al. 2024]. This means that InfiniBand can support up to $2^7 = 128$ addresses for the same destination. As introduced in Section 6.6.2, the LMC value used to implement the FORS routing solution is 5. Given that the topology includes 20 spines, even though not all switches necessarily use all the spines to reach a destination, we must consider the worst-case scenario to ensure the routing always works correctly. This means we need to consider that at least one switch might use all the spines to reach at least one destination server during the various phases of the schedule. Consequently, a minimum of 20 routes is required for each destination LID, which corresponds to $LMC = \lceil \log_2(20) \rceil = 5$. A LMC value of 5 provides $2^5 = 32$ routes for each destination.

A LMC value of 5 means that each switch contains 32 entries for each destination server in its routing table, plus one entry for each of the other switches in the topology. In the are FT(2;20,18), there are 38 switches. Consequently, with 326 servers in the LHCb DAQ network, each switch has $32 \times 326 + 37 =$ 10,469 entries. Since a modern Infiniband switch can hold up to 48,000 entries[technologies 2020], our solution remains applicable to larger topologies within an InfiniBand network. For instance, consider a fat-tree network, FT(2;36,36), with two layers, 36 leaf switches, and 36 spine switches, with a total of 1,296 servers. For this network, the number of entries required in the routing tables of the switches to implement FORS would be computed as $SP \times S + L + SP = 36 \times (36 \times 36) + 36 + 36 = 46,728$, where SP is the number of spine switches, S is the number of servers, and L is the number of leaf switches. However, to minimize the memory usage of the switches for the FORS routing and to improve compatibility with even larger InfiniBand networks, we developed an Mixed-Integer Linear Programming (MILP) model to optimize the LMC value.

The variables and constraints of the MILP model are consistent with those of the ILP model described in Section 6.5.1. The corresponding notations can be found in Table 6.3. The MILP model introduces the following additional variables:

- **Entry Variables** $e(l \in L, d \in S, i \in I)$**:** These binary variables indicate whether a specific entry exists in leaf switch $l \in L$ to join destination $d \in S$ using spine $i \in I$ as a path.
- **General Entry Variables** $ge(l \in L, d \in S)$**:** These integer variables represent the total number of entries in the routing table of leaf switch l to join destination d.
- **Maximum LMC Variable** $maxLMC$**:** This integer variable denotes the maximum LMC value required for routing. It represents the maximum number of entries needed in any switch to reach a destination.

The additional constraints are as follows:

1. **Definition of Entries:** If a spine is used by a leaf switch to join a destination, then the corresponding entry must exist in the routing table of the leaf switch. This is expressed as:

$$\forall l \in L,\, d \in S,\, d \notin S_l,\, i \in I,\, e(l,d,i) = \max(v(s \in S_l, d, i)),$$

   where max is the maximum function in Gurobi [experts 2023b].

2. **Definition of General Entries:** The general entry variable $ge(l,d)$ must equal the sum of all entries for joining destination d in the routing table of leaf switch l. This is formalized as:

$$\forall l \in L,\, d \in S,\, d \notin S_l,\, ge(l,d) = \sum_{i \in I} e(l,d,i).$$

3. **Definition of Maximum LMC:** The maximum LMC variable $maxLMC$ is defined as the maximum value of all general entry variables:

$$maxLMC = \max(ge(l \in L, d \in S)),$$

   where max is the maximum function in Gurobi [experts 2023b].

The optimization function minimize the value of the variable $maxLMC$. This model is a Mixed-Integer Linear Programming (MILP) model, not an Integer Linear Programming (ILP) model, because it uses the Gurobi max function[experts 2023b] in constraints 1 and 3, which introduces non-linearity.

Unfortunately, the computation time for this model becomes excessively long for network topologies with more than 12 switches and 32 servers. We were unable to obtain a solution for the LHCb DAQ network. To the best of our knowledge, further simplification of the model is not feasible. Therefore, until a more optimized approach for configuring multipath routing in InfiniBand is developed, the applicability of our solution will be constrained by the size of the network topology.

6.8 Conclusion

In this chapter, we present our Fault-adaptive Optimized Routing and Scheduling (FORS) solution, 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 traffic matrix. 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 an 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 spine switches, with throughput decreasing from 44 Tbps to 43 Tbps. As a result, FORS would be an interesting approach to address flapping links that can significantly impact network throughput, as shown in Chapter 4. We could deactivate a link that starts flapping and use FORS until the link is stable again.

Finally, we discuss the deployability of FORS on Infiniband networks and its limitations. Specifically, we propose a Mixed-Integer Linear Programming (MILP) model to minimize the number of entries in the routing tables of the switches. Given the limited memory available in switches, and although FORS remains applicable to large networks such as the studied DAQ network, this model offers a preliminary approach to reduce the memory usage required to deploy FORS on larger networks.

Chapter 7

Conclusion and Research Directions

In this thesis, we present an in-depth study of data acquisition (DAQ) networks and their fault tolerance. Our study focuses on the DAQ network of the Large Hadron Collider beauty (LHCb) experiment at the European Council for Nuclear Research (CERN). The LHCb DAQ network uses an all-to-all collective communication pattern with a fat-tree topology to make the best use of the bandwidth available and satisfy the DAQ system requirements. The routing algorithms used to route the all-to-all communications in the network are Infiniband routing algorithms specifically adapted to a fat-tree topology. We focus our work on a key network metric: the throughput. The LHCb data acquisition system lacks a hardware trigger for the initial and quick selection of collision events produced by the Large Hadron Collider (LHC). As a result, the DAQ network must handle the full collision rate. This constraint makes the studied DAQ network demanding in terms of throughput, making the network particularly challenging to optimize.

After covering the necessary background and related work relevant to our problem, we provided a detailed explanation of our three main contributions in the following three chapters: Chapters 4, 5, 6. In this chapter, we summarize these contributions and suggest perspectives and future directions to further improve our work.

7.1 Summary of the contributions

In this section, we summarize the key contributions presented in this thesis. We highlight the important information of each contribution and provide an overview.

7.1.1 Study of failures in the LHCb DAQ network

Our first contribution aims to understand the frequency, duration, and causes of various failures that can occur in a data acquisition (DAQ) network, as well as how these failures impact throughput depending on the routing algorithm used and the synchronization strategy. To address this, we present an analysis in Chapter 4 based on failure statistics observed over a two-month period during which the Large Hadron Collider (LHC) was operational, and consequently, the DAQ system was fully active. By monitoring these failures, we were able to extract information such as their duration and location in the network topology, allowing us to create statistics on frequency and duration.

To understand the causes of these failures, we investigated each of them to determine whether they were due to testing, dysfunction or maintenance. From this study, we concluded that failures in the DAQ network are frequent, with 2,605 failure events recorded during the data-taking period. Although most of these failures were brief with 88% lasted less than a minute and 97% lasted less than 10 minutes, when we consider flapping links, these failures can actually last for an extended period of time. During our measurement period, we observed that some links could flap, meaning they quickly transition between an up and a down state, for several consecutive minutes and even hours. When we combine flapping incidents on the same link and period as a single failure event, we were able to observe that over the 2605 observed failures, 1778 are attributed to flaps. Furthermore, when flaps are considered, only 24% of failures last less than a minute, and 60% last less than 10 minutes. This significantly increases the overall duration of failures.

After showing that failures, particularly flapping links, are not uncommon events in DAQ networks, we focused on measuring their impact on throughput. We first evaluated various Infiniband routing algorithms applicable to our DAQ network. We measured the throughput achieved by these routing algorithms under failure scenarios involving 0 to 10 consecutive failures in the network.

Our measurements exhibit that the default behavior of Infiniband on a fat-tree topology which is Ftree without failures switching to Min-Hop in case of failures, provides the best performance. We then analyzed the different results and explained why the other routing solutions do not perform the best for our network. From these measurements, we concluded that even a single failure significantly reduces throughput, dropping from approximately 144 Gbps per server to 87 Gbps, which represents a global throughput reduction from 46 Tbps to 28 Tbps, as the studied DAQ network interconnects 326 servers.

Finally, we demonstrated the impact of synchronization on throughput. We developed our own non-synchronized MPI application and measured the throughput achieved by this application compared to the synchronized version used in production. We found that the synchronized version performs better when there are no network failures, but when a failure occurs, the non-synchronized application outperforms the synchronized one.

This is due to the synchronized all-to-all application using the linear-shift pattern for scheduling communications and the Infiniband routing algorithm Ftree for routing them. The combination of this routing and scheduling solution achieves optimal performance when bandwidth is not reduced, as there is no congestion and each communication flow fully uses the links in the network. However, when bandwidth reduction occurs, congestion becomes inevitable, and synchronization no longer helps to prevent it. Additionally, synchronization causes servers not affected by the bandwidth reduction to be slowed down by those that are, as each server must wait for the others to finish their exchange before proceeding to the next phase.

As a result, the lack of synchronization makes better use of the remaining bandwidth during failures. Nevertheless, the non-synchronized all-to-all operation still experiences a low throughput of approximately 95.2 Gbps per server with one single failure compared to 144 Gbps when there are no failures, which represents a global throughput reduction from 46 Tbps to 31 Tbps.

7.1.2 Fault-Adaptive Scheduling Algorithm

As demonstrated in Chapter 4, failures occur frequently and significantly impact throughput, even when they are a few in the network. We initially focused on scheduling to understand why congestion arises even with a single failure and how to adapt to it. In Chapter 5, we show that the linear-shift pattern uses all network links at specific phases, which inevitably leads to congestion in the presence of failures. We then defined the concept of bandwidth reduction which allows us to understand the impact of failures on the routing of an all-to-all schedule.

We then consider an alternative scheduling pattern: Bandwidth-optimal. This approach optimizes the use of the bisection bandwidth at each phase, which should reduce congestion because not all links are used simultaneously. While [Prisacari *et al.* 2013b] formally describes the scheduling, we propose a well-defined algorithm to reproduce this scheduling and compare the bandwidth-optimal scheduling with the linear-shift. To conduct this comparison, we developed an algorithm to compute the amount of congestion created by each scheduling approach based on the network topology and the failure scenario, using the Infiniband Min-Hop routing to optimally distribute the load between links.

In this algorithm, we can also compute the time each scheduling approach takes to complete an all-to-all exchange, which allows us to measure their performance. We show that bandwidth-optimal scheduling does not outperform linear-shift scheduling, except in cases where only one failure occurs, which creates no congestion. We further demonstrate that, given the current number of phases, it is impossible for existing scheduling algorithms to avoid congestion during failures. Increasing the number of phases is necessary to avoid congestion, but this increase must be minimal as phases take time.

To address this issue, we propose a formula to compute the optimal number of phases based on the fat-tree topology and bandwidth reduction. We also introduce an algorithm combined with an Integer Linear Programming (ILP) model to adapt any scheduling algorithm to network failures without creating congestion. Our solution outperforms the bandwidth-optimal and linear-shift scheduling algorithms in terms of time required to complete an all-to-all exchange. However, our approach has limitations. The bandwidth reduction between servers must be uniform; otherwise, beyond a certain number of failures, the ILP model takes considerable time to adjust the schedule across additional phases without causing congestion. To handle all possible failure scenarios and avoid the computational delays of ILP, it is necessary to design a completely new scheduling algorithm.

7.1.3 Fault-adaptive Optimized Routing and Scheduling

To address the limitations of our contribution presented in Chapter 5, we focused on analyzing failure scenarios and discuss our findings in Chapter 6. We show that routing in a fat-tree topology can be reduced to selecting a spine (the switches at the top layer of a fat-tree topology) for each communication flow. By knowing the spine to be used for routing a communication flow in a fat-tree topology, we can define the path from the source to the spine and then from the spine to the destination. This characteristic of the fat-tree topology significantly simplifies the routing problem, as it reduces it to choosing a spine for each communication between two servers.

Then, we identified failure scenarios that create a non-uniform bandwidth reduction between switches. For instance, we demonstrate that if two leaf switches (the switches at the lowest layer of a fat-tree) experience link failures with different spines, the bandwidth reduction between these two leaf switches will be even greater compared to the bandwidth reduction they have with the other leaf switches. This scenario makes the routing more difficult to solve, as it must also be considered during scheduling to ensure that a routing solution exists for each failure scenario without congestion.

To address this issue, we propose a scheduling algorithm that respects an important property that we define: each leaf switch communicates at most once per phase with all other leaf switches. Therefore, regardless of whether the bandwidth reduction is uniform, as long as each leaf switch remains connected to all other leaf switches in the topology, a routing solution exists without creating congestion. Our scheduling algorithm is also optimal as it respects the minimum number of phases to perform an all-to-all exchange with bandwidth reduction, which is computed using the formula presented in Chapter 5.

To route the new scheduling algorithm, we propose a routing algorithm that handles failure scenarios where bandwidth reduction is uniform between switches. For cases where bandwidth reduction is non-uniform, we propose an Integer-Linear Programming (ILP) model to assign a spine for each communication flow.

We then evaluate our Fault-Adaptive Optimized Routing and Scheduling (FORS) solution in terms of throughput and compare it to the state of the art. FORS outperforms the alternatives. For example, with one link failure in the network, it achieves a throughput of 137 Gbps per server compared to 141 Gbps without failures, and it degrades gradually as the number of failures increases. As a reminder, the other scheduling approaches achieve, at best, 95 Gbps. This means that FORS achieves a global throughput of approximately 44 Tbps with one failure, while other approaches achieve, at best, 30 Tbps. Therefore, FORS would be a great alternative to address flapping links that make the network unstable for long periods of time, as shown in Chapter 4. We could simply deactivate a flapping link when it starts flapping and deploy FORS on the network until the link is stable again.

We also evaluate the computational time of FORS by simulating various random failure scenarios, ranging from one failure to the loss of half the links in the network. The minimum computation time is approximately 27 seconds, the maximum is around 37 seconds, and the average is about 30 seconds, making our routing solution applicable to the studied DAQ network.

Finally, we discuss the deployability of FORS on larger network topologies and the deployment constraints related to Infiniband limitations, which we hope will be addressed in future versions.

7.2 Perspectives

A first perspective to improve our work would be to study with more details the deployability of our Fault-Adaptive Optimized Routing and Scheduling (FORS) solution with Ethernet. It would be interesting to investigate how our routing algorithm could be implemented in an Ethernet network. In section 6.7.2, we explained in Chapter 6 that InfiniBand implements multipath routing using LID Mask Control (LMC), a mask that assigns multiple Local Identifiers (LIDs) to a destination server. With Ethernet, a comparable strategy could be to encode the spine to be used as a path for routing data directly into the IP address and applying a mask, similar to LMC, to know the routing path. Each destination would receive the same number of IP addresses as there are spines in the fat-tree topology. FORS would be less constrained on an Ethernet network, as it would allow encoding far more than the 128 addresses per destination, which is the limit in InfiniBand.

Currently, the network we studied uses InfiniBand for its flow control mechanisms and the 200 Gbps Infiniband network cards which was not possible with Ethernet as 200 Gbps Ethernet network cards were not available at the time the network was designed. However, Ethernet is more standardized and allows the use of other routing strategies, such as Equal-Cost Multi-Path (ECMP). ECMP spreads the traffic on multiple paths. Here, because our different flows are roughly of equivalent size, ECMP could help us reduce network congestion.

In general, the deployability of FORS should be more thoroughly studied on InfiniBand networks, particularly by developing our own routing engine to deploy FORS dynamically. Currently, FORS can not be automatically generated in the event of failures because we need to create a file containing the routing tables, which includes routes to join a destination server at each phase and for each switch while avoiding failed ports. With the current OpenSM implementation, in case of a failed port for an entry in the routing table, OpenSM needs to load the new routing tables from our configuration file. In the meantime, packets are dropped. To address this, we would need to develop our own routing engine that can compute our routing and scheduling solution and generate the associated routing tables. This would allow the controller to dynamically switch to our solution in the event of a failure, similar to how Ftree defaults to Min-Hop in the event of failures without restarting the OpenSM controller.

Once FORS is deployed on the LHCb DAQ network, another future work would be to evaluate its fail-over performance. To be more precise, it would be interesting to measure the time taken from the occurrence of a failure to the point when the network stabilizes with the FORS solution. Additionally, we should track the frequency and duration of FORS utilization over several months, and then compute the amount of data preserved by using FORS. This could be achieved by comparing the throughput achieved with FORS to the throughput of the currently deployed routing and scheduling solution, using the measurements presented in Chapter 4.

An interesting area for further research would be to formally define the complexity of the routing problem when bandwidth reduction is non-uniform. To address this, we proposed an Integer Linear Programming (ILP) model, as described in Section 6.5. Although we attempted to develop a routing algorithm to solve this problem, we were unsuccessful. We believe this challenge arises because our problem could be a Multi-Commodity Flow Problem, specifically an Unsplittable Flow Problem (UFP). In UFP, each communication flow between a source and a destination in a network must use a single path and cannot be split between multiple paths while respecting capacity constraint on the links. This problem is known to be NP-hard, and formally defining our routing problem with non-uniform bandwidth reduction as a UFP could provide deeper insights into its complexity.

Constraint Programming (CP) is a technique used to find solutions that satisfy a set of constraints, rather than optimizing an objective function as in linear programming. CP is particularly effective for solving combinatorial problems where the constraints are logical [Google 2024]. For instance, it performs well when variables must be all different or when a constraint requires that if one variable is equal to 1, another must be equal to 0. However, the problem of routing communications with a non-uniform bandwidth reduction is more naturally expressed using linear constraints, which is why we use Integer Linear Programming (ILP) as described in Section 6.5. Logical constraints alone may not adequately capture the complexity of the routing problem. While it is possible to express our routing problem purely with logical constraints, to our knowledge, doing so requires the introduction of additional variables and constraints to replicate the behavior of linear constraints. This may not only complicates the model but also slows down the solver. We attempted to solve the routing problem with a non-uniform bandwidth reduction using logical constraints suitable for CP. We used the open-source CP-SAT solver from Google's OR-Tools[Google 2023]. However, a solution could not be found within a reasonable time for the topology of the studied DAQ network. Nevertheless, this approach may need further investigation.

Another perspective for future work would be to adapt our scheduling solution to other collective operations such as AllGather. As presented in Chapter 3, while AllGather uses the same bandwidth as all-to-all, it involves a different number of exchanged messages, with all-to-all requiring more messages to be exchanged in the network. Therefore, our scheduling solution could likely be adapted to these collective operations as they are less challenging to optimize in the event of failures. Furthermore, the property in our scheduling to address non-uniform bandwidth reduction and the routing solution can also be applied to these operations.

Finally, a last perspective would be to study how FORS could be adapted to other topologies such as HyperX and AB Fat-Tree, which are discussed in Chapter

3. The AB Fat-Tree topology is part of the Fault-Tolerant Engineered Network (F10) solution designed to enhance fault tolerance. This topology increases the number of paths between leaf switches in the event of a failure by breaking the symmetry of a standard fat-tree topology. In the LHCb DAQ network, the AB fat-tree topology is not particularly advantageous, as there are already the maximum number of paths between all leaf switches due to the two-layer structure of the fat-tree topology. However, for enhancing fault tolerance in fat-tree topologies with more than two layers, the AB Fat-Tree topology is an interesting solution, and FORS could be adapted to accommodate this topology. We could evaluate the AB Fat-Tree topology in terms of fault tolerance, for instance, by determining whether the ILP model for routing solutions remains necessary in cases of non-uniform bandwidth reduction. Given that the AB Fat-Tree topology provides more paths between leaf switches, our simpler routing algorithm for uniform bandwidth reduction might suffice. Additionally, we could adapt FORS to HyperX topologies and assess whether a simpler routing solution would be adequate for traffic routing, considering that the HyperX topology offers many redundant paths.

UNIVERSITE OF STRASBOURG

ECOLE DOCTORALE MSII

MATHEMATIQUES, SCIENCES DE L'INFORMATION ET DE L'INGENIEUR

Laboratoire ICube, équipe réseaux

T H È S E présentée par :

Eloïse Stein

soutenue le : 21 Octobre 2024

pour obtenir le grade de :

Docteur de l'Université de Strasbourg

Discipline/Spécialité : Informatique

Ordonnancement et routage intelligent pour les réseaux d'acquisition de données

THÈSE dirigée par :

Prof. Cristel Pelsser	-	Université de Strasbourg Université catholique de Louvain (UCLouvain)
Prof. Thomas Noel	-	Université de Strasbourg

RAPPORTEURS :

Prof. Holger Fröning	-	Université d'Heidelberg
Prof. Piero Vicini	-	Institut national de physique nucléaire (INFN) (INFN)

AUTRES MEMBRES DU JURY :

Prof. Géraldine Texier	-	École nationale supérieure Mines-Télécom Atlantique (IMT)
Prof. Stefano Secci	-	Conservatoire National des Arts et Métiers (Cnam)

Résumé

Les grands instruments scientifique sont généralement composés de dizaines de milliers de capteurs et d'un système d'acquisition de données (DAQ) dont l'objectif est de collecter les fragments de données de chaque capteur et d'assembler tous les fragments en un seul ensemble de données. Ce processus, appelé Event Building, implique un échange collectif de type all-to-all entre un ou plusieurs ordinateurs interconnectés en réseau. Cependant, le trafic associé à l'Event Building a tendance à créer de la congestion en cas de panne en raison de la nature de l'échange all-to-all, qui nécessite d'utiliser presque toute la bande-passante disponible sur le réseau. La congestion peut entraver gravement les performances du système DAQ et entraîner la perte de données expérimentales très précieuses. Cette thèse présente des approches visant à minimiser la congestion du réseau en cas de panne, en utilisant le réseau DAQ du détecteur LHCb (Large Hadron Collider beauty) comme étude de cas.

Abstract

Large scientific instruments are typically composed of tens of thousands of sensors and a data acquisition (DAQ) system whose objective is to collect fragments of data from each sensor and to assemble all the fragments corresponding to each collision event into a single data set. This process, called Event Building, involves an all-to-all collective exchange between one or more network computers. However, the traffic associated with Event Building tends to create congestion in the event of a failure due to the nature of the all-to-all exchange, which requires using almost all available network bandwidth. Congestion can severely degrade the performance of the DAQ system and lead to the loss of highly valuable experimental data. This thesis presents approaches to minimizing network congestion in the event of failures, using the DAQ network of the LHCb (Large Hadron Collider beauty) detector as a case study.

Remerciements

À l'approche de la fin de ce parcours de trois ans, je pense à toutes les personnes qui l'ont partagé avec moi. Ces années ont été remplies de défis et de croissance, et je n'aurais pas pu les traverser sans le soutien, les encouragements et la bienveillance de tant de personnes. Bien qu'il soit impossible d'exprimer pleinement la gratitude que je ressens envers tous ceux qui ont fait partie de cette expérience, je tiens à prendre un moment pour remercier chaque personne qui a contribué à ce voyage.

Tout d'abord, je tiens à exprimer mes sincères remerciements à ma directrice de thèse, Cristel Pelsser. Votre guidance tout au long de cette recherche a été inestimable, et elle n'aurait pas atteint ce stade sans votre expertise et vos conseils. J'ai vraiment apprécié nos discussions au cours de ces trois dernières années, votre compréhension de ce sujet de recherche, qui était nouveau pour nous deux, ainsi que votre intérêt pour la programmation linéaire et les suggestions qui m'ont grandement aidée à développer les solutions présentées dans cette thèse. Enfin, merci pour votre incroyable soutien et vos encouragements, sans lesquels cette thèse n'aurait pas pu être achevée. Vous êtes une directrice de thèse exceptionnelle et tout étudiant se lançant dans ce parcours, comme je l'ai fait, aura beaucoup de chance de vous avoir pour le guider.

Je tiens également à remercier les membres de mon jury qui ont accepté d'évaluer ma thèse. Merci à Holger Fröning, Piero Vicini, Géraldine Texier et Stefano Secci. J'espère sincèrement que vous avez trouvé mon travail, qui a nécessité beaucoup de temps et d'efforts de ma part et de celle de toutes les personnes qui ont contribué, à la fois intéressant et pertinent.

Ensuite, un grand merci à Quentin Bramas, sans qui une partie importante de ce travail n'aurait pas été possible. Vos conseils précieux et votre expertise en mathématiques ont grandement contribué à résoudre certains des principaux défis présentés dans cette thèse. Je vous suis profondément reconnaissante pour vos contributions, qui ont joué un rôle crucial dans l'avancement de cette recherche.

Je remercie Pierre Schaus pour les précieuses remarques sur les défis d'optimisation auxquels j'ai été confrontée pendant cette thèse.

Je tiens également à remercier toute l'équipe en ligne du CERN pour leurs contributions à ce travail et pour avoir apporté leur expertise lorsque cela était nécessaire. En particulier, je suis reconnaissante à Flavio Pisani et Tommaso Colombo, qui ont consacré une journée et une nuit entières à tester le travail présenté dans cette thèse avec moi. Je souhaite exprimer mes remerciements à Tommaso Colombo pour sa supervision tout au long de ce projet. Mes remerciements vont aussi à Alberto Perro, Pierfrancesco Cifra et Konstantinos Stavropoulos pour les discussions enrichissantes que nous avons eues autour d'une pizza. J'apprécie les conversations intéressantes avec Mauricio Feo, qu'elles soient académiques ou non, et pour ses encouragements. Enfin, je veux remercier Clara Gaspar et Niko Neufeld, les responsables de cette équipe, qui accomplissent un travail incroyable chaque jour et m'ont soutenue, m'aidant à rester motivée et concentrée sur mes futures opportunités.

Je remercie Denis, sans qui rien de tout cela ne serait arrivé, ainsi que pour ses conseils, en lui souhaitant bon courage pour la fin de son périple.

Je tiens à remercier Jeanne et Thierry pour leur soutien, leurs conseils et leur présence. Leur force de travail et de caractère m'ont toujours motivée à aller de l'avant.

Je remercie également Adèle, dont la détermination, la joie de vivre et la volonté auront été une source d'inspiration dans mon travail.

Je tiens également à exprimer ma profonde gratitude à mes parents, Yolande et Fabrice, pour leur soutien et leur amour tout au long de ce parcours.

Enfin, et surtout, un grand merci à mon partenaire, Hugo. Il est difficile d'exprimer par des mots à quel point ta présence, ton amour et ton soutien ont compté pour moi tout au long de ce parcours. Tu ne le croiras peut-être pas, mais sans toi, ce travail n'aurait jamais abouti. Tes encouragements et ta gentillesse ont été essentiels pour rendre cette thèse possible. Je n'aurais pas pu terminer ce doctorat sans ton incroyable soutien, et je t'en serai toujours profondément reconnaissante.

Ce travail a été principalement financé par le Conseil européen pour la recherche nucléaire (CERN) et partiellement financé par une bourse de la Silicon Valley Foundation (CG1318167).

7.3 Introduction

Les systèmes d'acquisition de données (DAQ) jouent un rôle crucial dans la collecte de données scientifiques [Belyaev *et al.* 2017, Jereczek *et al.* 2015, Bawej *et al.* 2015]. Généralement composés d'un ensemble diversifié de capteurs, les systèmes DAQ capturent de grandes quantités de données. Ces systèmes sont largement utilisés dans divers domaines, notamment la recherche scientifique, tels que l'aérospatiale [Borrill *et al.* 2015, Dorelli *et al.* 2022, Update 2011], les soins de santé [Liu *et al.* 2023, Leung *et al.* 2020] et la physique [Belyaev *et al.* 2017, Jereczek *et al.* 2015, Bawej *et al.* 2015]. Par exemple, les systèmes DAQ de l'Organisation européenne pour la recherche nucléaire (CERN)[Jereczek *et al.* 2015, Bawej *et al.* 2015] traitent des dizaines d'exaoctets de données chaque année[LHCb Collaboration 2014, LHCb Collaboration 2020], contribuant de manière significative aux avancées dans le domaine de la recherche en physique[CERN 2019]. De tels systèmes sont déployés dans les expériences menées le long du Grand collisionneur de hadrons (LHC) au CERN pour collecter des données fragmentées provenant de différents capteurs et les assembler afin de reconstruire chaque événement de collision de particules. Ce processus est connu sous le nom de "construction d'événements" (Event Building).

La construction d'événements pour les expériences menées sur de grands collisionneurs repose généralement sur un réseau à haut débit et à faible latence interconnectant des serveurs. Les données échangées sont produites en temps réel par de grands instruments scientifiques tels que le LHC. Le LHC au CERN accélère des particules à des énergies allant jusqu'à 6,8 TeV et les fait entrer en collision à des énergies atteignant 13,6 TeV. De nombreux capteurs enregistrent divers aspects des collisions résultantes, également appelées "événements". Chaque capteur est connecté à un serveur qui reçoit ses données. Pour synthétiser ces fragments de données en une représentation unifiée de chaque événement, chaque serveur échange ses fragments de données avec tous les autres via le réseau. La matrice de trafic réseau qui en résulte est une succession continue d'échanges de type "tous vers tous". Comme les événements sont produits en continu dans le LHC, les serveurs doivent reconstruire les événements rapidement. Un retard dans la reconstruction peut surcharger la mémoire tampon des serveurs avec des données, entraînant une congestion et la perte potentielle de données importantes. Par conséquent, le débit est la métrique réseau centrale examinée dans cette thèse.

L'échange collectif de type "tous vers tous" exige des ressources réseau significatives, telles que la bande passante. Pour optimiser l'utilisation de la bande passante disponible dans le réseau, la capacité totale d'un lien est utilisée pour chaque transmission entre serveurs. Si deux flux de communication utilisent le même lien, une congestion se produit, car les deux transmissions doivent partager la capacité du lien. Par conséquent, l'utilisation de la bande passante dans le réseau DAQ est proche de la capacité maximale, ce qui rend l'échange "tous vers tous" très sensible aux défaillances. Comme le souligne la littérature [Gill *et al.* 2011], les pannes de liens dans les réseaux à haut débit, tels que ceux utilisés pour les communications collectives entre serveurs dans les réseaux DAQ, sont fréquentes.

Les systèmes DAQ s'appuient souvent sur des applications de calcul haute performance (HPC) pour l'analyse en temps réel et le traitement efficace de volumes importants de données. L'optimisation des applications HPC fait l'objet de nombreuses recherches, repoussant les limites des composants matériels[Wu *et al.* 2023, Wang *et al.* 2023a], des architectures[Chirkov & Wentzlaff 2023, Huang *et al.* 2023, Contini *et al.* 2023] de calcul et des réseaux[Feng *et al.* 2023]. Un manque d'optimisation peut entraîner une perte de performance significative, empêchant la réalisation de découvertes scientifiques importantes. Bien que l'optimisation des échanges collectifs, tels que l'échange "tous vers tous", dans les applications HPC soit largement abordée et étudiée [Prisacari *et al.* 2013b, Prisacari *et al.* 2013a, Al-Fares *et al.* 2010, Izzi & Massini 2020, Zahavi *et al.* 2009, Peng *et al.* 2022, Izzi & Massini 2022, Izzi & Massini 2023], à notre connaissance, il n'existe aucune proposition visant à optimiser ces échanges collectifs en cas de défaillances réseau impliquant une réduction de bande passante, malgré la fréquence élevée de ces défaillances [Singh *et al.* 2021]. Comme nous le démontrons dans cette thèse, les performances réseau sont actuellement fortement impactées en cas de réduction de la bande passante. Résoudre ce problème est un défi, car cela implique d'adapter l'ordonnancement et le routage de ces échanges collectifs à la bande passante résiduelle du réseau en situation de défaillance.

Les opérations collectives, en général, sont également de plus en plus utilisées dans l'apprentissage automatique[Sergeev & Del Balso 2018, Zhao *et al.* 2024a, Zhou *et al.* 2023, Wang *et al.* 2023b], notamment AllGather, AllReduce ou ReduceScatter [Nvidia 2020]. Cependant, elles diffèrent de l'échange collectif "tous vers tous". Notre problématique est spécifique aux systèmes DAQ.

L'échange "tous vers tous" est une communication collective très exigeante en termes de bande passante, car tous les serveurs du réseau DAQ doivent échanger des données avec tous les autres. Si tous les serveurs envoient des données vers une même destination simultanément, les liens vers cette destination deviennent congestionnés. Une stratégie typique pour résoudre ce problème consiste à répartir les communications de chaque serveur dans le temps, ce qui signifie que l'échange est segmenté en phases distinctes. Dans cette approche, l'échange collectif "tous vers tous" est synchronisé, ce qui signifie que l'application DAQ garantit que tous les serveurs terminent leurs échanges de données avant de passer à la phase suivante. Cette approche synchronisée permet d'atteindre un débit élevé, en particulier dans les systèmes avec des débits proches de 100% des capacités des liens [Pisani *et al.* 2023a].

7.3.1 Contributions

Dans cette thèse, je présente mes contributions aux solutions de routage et d'ordonnancement visant à améliorer la tolérance aux pannes dans les réseaux DAQ. De plus, je démontre, à travers des mesures, l'importance de traiter ce problème, compte tenu de la sensibilité des réseaux DAQ aux défaillances.

Tout d'abord, nous proposons d'étudier les pannes de liens dans le réseau DAQ de l'expérience Large Hadron Collider beauty (LHCb) au CERN. Nous analysons les pannes réseau observées sur une période de deux mois, durant laquelle le LHC était pleinement actif et des données étaient échangées dans le réseau DAQ. Plus précisément, nous présentons des statistiques sur la durée, la fréquence et les causes sous-jacentes des pannes de liens réseau pour démontrer que ces pannes sont fréquentes et peuvent durer longtemps. Ces statistiques justifient notre problématique.

Ensuite, nous présentons une analyse de divers algorithmes de routage adaptés au réseau étudié. Plus précisément, nous évaluons les performances du routage adaptatif par rapport à l'algorithme de routage actuellement utilisé dans le réseau DAQ de l'expérience LHCb, dans différents scénarios de panne. Nos mesures mettent en évidence le débit atteint par chaque algorithme de routage sur le réseau étudié. Nous constatons que les pannes de liens peuvent dégrader significativement les performances, car l'utilisation de la bande passante dans le réseau DAQ est proche de la capacité maximale. Une seule panne entraîne une congestion et réduit le débit total d'environ 46 Tbps à 30 Tbps, même avec la meilleure solution de routage disponible. Nous proposons également d'évaluer les deux approches d'ordonnancement de l'échange collectif "tous vers tous" : non-synchronisé et synchronisé. Dans un échange "tous vers tous" synchronisé, tous les serveurs attendent que les autres aient terminé leur échange de données avant de passer à la phase suivante. C'est l'approche actuellement utilisée dans le réseau DAQ étudié. Une approche alternative consiste en un échange "tous vers tous" non-synchronisé, plus simple, où le réseau gère la congestion. Ces deux approches n'ont jamais été comparées dans le contexte des pannes de liens réseau, ce que nous proposons dans le Chapitre 4. Dans cette thèse, nous ne prenons pas en compte les pannes entre les commutateurs réseau et les serveurs, car ces pannes entraînent une déconnexion totale des serveurs affectés du réseau. Ces serveurs ne peuvent plus participer à l'échange "tous vers tous", et leur exclusion n'impacte pas la bande passante nécessaire pour terminer l'échange entre les serveurs restants. Enfin, nous proposons quelques recommandations de conception issues de la comparaison du débit atteint par les applications "tous vers tous" synchronisées et non synchronisées en présence de pannes réseau. Cependant, aucune de ces approches d'ordonnancement et de routage ne permet d'utiliser efficacement la bande passante restante dans le réseau lorsque des pannes surviennent. Ces contributions sont abordées dans le Chapitre 4 et ont abouti à une publication [Stein *et al.* 2024].

Comme nous démontrons l'importance de trouver une solution d'ordonnancement et de routage qui assure une dégradation plus progressive du débit, nous menons une étude détaillée des algorithmes d'ordonnancement existants dans la littérature. Nous montrons que ces algorithmes ne sont pas tolérants aux pannes, car le nombre de phases dans un échange "tous vers tous" synchronisé est insuffisant pour éviter la congestion. Pour y remédier, nous introduisons le concept de réduction de bande passante, qui définit plus précisément comment les pannes affectent le routage dans un plan d'ordonnancement "tous vers tous" et nous permet de dériver une formule pour calculer la borne inférieure du nombre de phases nécessaires pour un routage sans congestion, rendant ainsi le nombre de phases optimal. Ensuite, nous proposons un algorithme, accompagné d'un modèle de Programmation Linéaire Entière (ILP), pour adapter les schémas d'ordonnancement "tous vers tous" existants afin de gérer les scénarios de panne sur une topologie fat-tree. Enfin, nous évaluons notre contribution et mettons en évidence ses limites. Cette contribution a donné lieu à une publication[Stein *et al.* 2023].

Pour remédier à ces limitations, nous proposons une étude plus complète de divers scénarios de pannes dans le réseau, en mettant l'accent sur les défis liés à l'ordonnancement et au routage afin de prévenir la congestion. Ensuite, nous présentons une solution de Routage et Ordonnancement Optimisés et Adaptatifs aux Pannes (FORS) pour maintenir un débit élevé dans les échanges "tous vers tous" malgré les goulots d'étranglement introduits par les pannes de liens réseau. FORS se compose d'un algorithme permettant d'adapter l'ordonnancement des communications pour l'échange collectif "tous vers tous" en cas de pannes de liens. De plus, FORS comprend une solution de routage semi-algorithmique, combinant un algorithme de calcul de routes pour les scénarios de pannes simples avec un modèle de Programmation Linéaire Entière (PLE) conçu pour traiter des combinaisons de pannes plus complexes. L'objectif de ces deux algorithmes est de fournir des chemins sans congestion entre les serveurs du réseau en fonction du scénario de panne donné. Nous démontrons l'applicabilité et les performances de notre solution sur un réseau DAQ réel et opérationnel au CERN, utilisant le calcul haute performance (HPC) pour traiter de grands volumes de données scientifiques. Nous comparons notre proposition avec les approches actuellement déployées. Cette contribution a donné lieu à une publication.

7.3.2 Aperçu

Cette thèse se compose de sept chapitres. Le Chapitre 2 fournit les éléments de base nécessaires pour comprendre nos contributions. Nous commençons par introduire la topologie fat-tree et les deux variantes que nous considérons. Ensuite, nous discutons de l'échange collectif "tous vers tous" et de l'algorithme d'ordonnancement linéaire par décalage, largement utilisé dans les réseaux InfiniBand. Nous donnons ensuite un aperçu du réseau DAQ de l'expérience LHCb et du processus de construction des événements (Event Building). Nous présentons plus en détail l'application de l'échange collectif "tous vers tous" dans le processus de construction des événements et l'utilisation de la technologie InfiniBand dans la topologie fat-tree du réseau DAQ.

Dans le Chapitre 3, nous passons en revue les travaux connexes pertinents pour notre recherche, y compris diverses opérations collectives existantes autres que l'échange "tous vers tous". Nous présentons également différentes stratégies de routage, telles que les stratégies adaptatives, non-sensibles et tenant compte de l'ordonnancement. Enfin, nous introduisons d'autres topologies réseau telles que Dragonfly, largement utilisée dans les réseaux HPC, ainsi que des topologies adaptées à la demande. Nous présentons les topologies HyperX et F10 qui sont pertinentes pour notre problème, car elles pourraient améliorer la tolérance aux pannes en augmentant le nombre et la diversité des chemins disponibles entre les commutateurs.

Dans le Chapitre 4, nous présentons nos statistiques sur les pannes réseau qui se sont produites dans le réseau étudié pendant les mois de mars et avril 2024. Nous présentons ensuite des mesures du débit atteint par différents algorithmes de routage InfiniBand pertinents pour notre réseau, ainsi que des données montrant les effets de la synchronisation de l'échange "tous vers tous" en termes de scalabilité et en cas de pannes. Nous concluons ce chapitre par des recommandations de conception.

Dans le Chapitre 5, nous étudions plus en détail divers algorithmes d'ordonnancement et démontrons leur inadéquation en cas de pannes dues à la congestion inévitable, car aucune solution de routage sans congestion n'existe avec le nombre de phases proposé. Nous expliquons ensuite comment nous calculons le nombre optimal de phases en fonction de la réduction de bande passante causée par les pannes réseau. Enfin, nous proposons une première solution d'ordonnancement qui peut adapter n'importe quel modèle d'ordonnancement "tous vers tous" en cas de pannes, en ne tenant compte que des scénarios de pannes simples.

Dans le Chapitre 6, nous menons une analyse plus approfondie des différents scénarios de pannes et de leurs conséquences en termes de réduction de bande passante. Nous proposons ensuite un algorithme d'ordonnancement qui prend plus efficacement en compte le routage, garantissant qu'une solution de routage faisable sans congestion existe pour tous les scénarios de pannes. De plus, nous présentons un algorithme de routage associé à un modèle de Programmation Linéaire Entière (PLE) pour traiter tous les scénarios de pannes possibles tant que les commutateurs réseau directement connectés aux serveurs restent entièrement connectés. Enfin, nous évaluons la performance et l'applicabilité de notre solution.

Dans le Chapitre 7, nous concluons en résumant nos diverses contributions et leurs résultats, ainsi qu'en suggérant des perspectives de recherche futures.

7.4 Conclusion et perspectives de recherche

Dans cette thèse, nous présentons une étude approfondie des réseaux d'acquisition de données (DAQ) et de leur tolérance aux pannes. Notre étude se concentre sur le réseau DAQ de l'expérience Large Hadron Collider beauty (LHCb) au Conseil Européen pour la Recherche Nucléaire (CERN). Le réseau DAQ de LHCb utilise un schéma de communication collective de type tout-à-tout avec une topologie en fat-tree afin d'exploiter au mieux la bande passante disponible et de répondre aux exigences du système DAQ. Les algorithmes de routage utilisés pour acheminer ces communications tout-à-tout sont des algorithmes de routage Infiniband spécifiquement adaptés à une topologie en fat-tree.

Nous concentrons notre travail sur une métrique clé du réseau : le débit. Le système d'acquisition de données de LHCb ne dispose pas de déclencheur matériel permettant une sélection initiale et rapide des événements de collision produits par le Grand Collisionneur de Hadrons (LHC). Par conséquent, le réseau DAQ doit gérer l'intégralité du taux de collision. Cette contrainte impose des exigences élevées en matière de débit, rendant l'optimisation du réseau particulièrement complexe.

Après avoir présenté le contexte nécessaire et les travaux connexes en lien avec notre problématique, nous détaillons nos trois principales contributions dans les chapitres suivants : Chapitres 4, 5, 6. Dans ce chapitre, nous résumons ces contributions et proposons des perspectives ainsi que des orientations futures pour améliorer davantage notre travail.

7.4.1 Résumé des contributions

Dans cette section, nous résumons les principales contributions présentées dans cette thèse. Nous mettons en avant les éléments clés de chaque contribution et en fournissons une vue d'ensemble.

7.4.1.1 Étude des pannes dans le réseau DAQ de LHCb

Notre première contribution vise à comprendre la fréquence, la durée et les causes des différentes pannes pouvant survenir dans un réseau d'acquisition de données (DAQ), ainsi que leur impact sur le débit en fonction de l'algorithme de routage utilisé et de la stratégie de synchronisation adoptée. Pour cela, nous présentons dans le Chapitre 4 une analyse basée sur les statistiques de pannes observées sur une période de deux mois durant laquelle le Grand Collisionneur de Hadrons (LHC) était en fonctionnement et, par conséquent, le système DAQ pleinement actif. En surveillant ces pannes, nous avons pu extraire des informations telles que leur durée et leur emplacement dans la topologie du réseau, ce qui nous a permis d'établir des statistiques sur leur fréquence et leur durée.

Afin d'identifier les causes de ces pannes, nous avons examiné chaque incident pour déterminer s'il était dû à des tests, un dysfonctionnement ou une maintenance. Cette étude nous a permis de conclure que les pannes dans le réseau DAQ sont fréquentes, avec 2 605 événements enregistrés durant la période de collecte des données. Bien que la majorité de ces pannes aient été brèves avec 88% d'entre elles durant moins d'une minute et 97% moins de 10 minutes, la prise en compte des liens instables (liens qui flappent) révèle une durée effective plus longue. Pendant notre période de mesure, nous avons observé que certains liens pouvaient fluctuer rapidement entre un état actif et inactif pendant plusieurs minutes, voire plusieurs heures. En regroupant ces fluctuations sur un même lien en un seul événement de panne, nous avons constaté que, parmi les 2 605 pannes observées, 1 778 étaient dues à des fluctuations de lien. En considérant ces fluctuations, seulement 24% des pannes durent moins d'une minute, et 60% moins de 10 minutes, ce qui augmente significativement la durée globale des pannes.

Après avoir montré que les pannes, en particulier celles causées par des liens instables, ne sont pas des événements rares dans les réseaux DAQ, nous avons évalué leur impact sur le débit. Nous avons d'abord analysé différents algorithmes de routage Infiniband applicables à notre réseau DAQ et mesuré le débit atteint sous divers scénarios de pannes impliquant de 0 à 10 pannes consécutives dans le réseau.

Nos mesures montrent que le comportement par défaut d'Infiniband sur une topologie en fat-tree, à savoir l'utilisation de Ftree en l'absence de panne, avec un basculement vers Min-Hop en cas de panne, offre les meilleures performances. Nous avons ensuite analysé les différents résultats et expliqué pourquoi les autres solutions de routage ne sont pas les plus adaptées à notre réseau. Nous avons conclu que même une seule panne entraîne une réduction significative du débit, passant d'environ 144 Gbps par serveur à 87 Gbps, soit une réduction du débit global de 46 Tbps à 28 Tbps, le réseau DAQ étudié interconnectant 326 serveurs.

Enfin, nous avons démontré l'impact de la synchronisation sur le débit. Nous avons développé notre propre application MPI non synchronisée et mesuré le débit obtenu par cette application par rapport à la version synchronisée utilisée en production. Nos résultats montrent que la version synchronisée offre de meilleures performances en l'absence de pannes, mais qu'en cas de panne, l'application non synchronisée surpasse la version synchronisée.

Ce phénomène s'explique par le fait que l'application tout-à-tout synchronisée utilise le schéma de communication en décalage linéaire (linear-shift) et que l'algorithme de routage Ftree d'Infiniband est utilisé pour les acheminer. Cette combinaison de routage et d'ordonnancement permet d'atteindre des performances optimales tant que la bande passante n'est pas réduite, car il n'y a pas de congestion et chaque flux de communication exploite pleinement les liens du réseau. Cependant, en présence d'une réduction de la bande passante, la congestion devient inévitable et la synchronisation ne permet plus de l'éviter. De plus, la synchronisation entraîne un ralentissement des serveurs non affectés par la réduction de bande passante, car ils doivent attendre que les autres terminent leurs échanges avant de passer à la phase suivante.

Ainsi, l'absence de synchronisation permet une meilleure exploitation de la bande passante restante en cas de panne. Néanmoins, même avec un fonctionnement tout-à-tout non synchronisé, le débit reste limité : il atteint environ 95,2 Gbps par serveur en présence d'une seule panne, contre 144 Gbps en l'absence de panne, ce qui représente une réduction du débit global de 46 Tbps à 31 Tbps.

7.4.1.2 Algorithme d'Ordonnancement Adaptatif aux Pannes

Comme démontré au Chapitre 4, les pannes sont fréquentes et ont un impact significatif sur le débit, même lorsqu'elles sont peu nombreuses dans le réseau. Nous nous sommes d'abord concentrés sur l'ordonnancement afin de comprendre pourquoi la congestion survient même avec une seule panne et comment s'y adapter. Dans le Chapitre 5, nous montrons que le schéma d'ordonnancement en décalage linéaire (linear-shift) utilise tous les liens du réseau à des phases spécifiques, ce qui conduit inévitablement à une congestion en présence de pannes. Nous avons ensuite défini le concept de réduction de bande passante, qui nous permet de comprendre l'impact des pannes sur l'acheminement d'un schéma de communication de type all-to-all.

Nous avons alors étudié une alternative : l'ordonnancement Bandwidth-optimal. Cette approche optimise l'utilisation de la bande passante de bisection à chaque phase, ce qui devrait réduire la congestion, car tous les liens ne sont pas utilisés simultanément. Bien que l'ordonnancement soit formellement décrit[Prisacari *et al.* 2013b], nous proposons un algorithme bien défini pour le reproduire et le comparer au schéma en décalage linéaire. Pour mener à bien cette comparaison, nous avons développé un algorithme permettant de calculer la congestion créée par chaque approche d'ordonnancement en fonction de la topologie du réseau et du scénario de panne, en utilisant l'algorithme de routage Min-Hop d'Infiniband afin de répartir la charge de manière optimale entre les liens. Cet algorithme nous permet également de calculer le temps nécessaire à chaque approche pour compléter un échange all-to-all, ce qui nous permet d'évaluer leur performance. Nous montrons que l'ordonnancement Bandwidth-optimal ne surpasse pas l'ordonnancement en décalage linéaire, sauf dans les cas où une seule panne est présente, car elle ne génère alors aucune congestion. Nous démontrons ensuite qu'avec le nombre actuel de phases, il est impossible pour les algorithmes d'ordonnancement existants d'éviter la congestion en cas de panne. Augmenter le nombre de phases est nécessaire pour éviter la congestion, mais cette augmentation doit être minimale, car chaque phase prend du temps.

Pour répondre à ce problème, nous proposons une formule permettant de calculer le nombre optimal de phases en fonction de la topologie en fat-tree et de la réduction de bande passante. Nous introduisons également un algorithme combiné à un modèle de Programmation Linéaire en Nombres Entiers (ILP) afin d'adapter n'importe quel algorithme d'ordonnancement aux pannes du réseau sans créer de congestion. Notre solution surpasse les algorithmes d'ordonnancement Bandwidth-optimal et en décalage linéaire en termes de temps nécessaire pour effectuer un échange all-to-all. Cependant, notre approche présente des limitations. La réduction de bande passante entre serveurs doit être uniforme, sinon, au-delà d'un certain nombre de pannes, le modèle ILP met un temps considérable à ajuster l'ordonnancement sur des phases supplémentaires sans créer de congestion. Pour gérer tous les scénarios de panne possibles et éviter les délais de calcul liés à l'ILP, il est nécessaire de concevoir un algorithme d'ordonnancement entièrement nouveau.

7.4.1.3 Routage et Ordonnancement Optimisés et Adaptatifs aux Pannes

Pour surmonter les limites de notre contribution présentée au Chapitre 5, nous avons analysé différents scénarios de panne, comme détaillé au Chapitre 6. Nous montrons que le routage dans une topologie en fat-tree peut être réduit au choix d'une spine (les commutateurs de la couche supérieure de la topologie en fat-tree) pour chaque flux de communication. En connaissant la spine utilisée pour router un flux dans un fat-tree, nous pouvons définir le chemin de la source vers la spine, puis de la spine vers la destination. Cette propriété simplifie considérablement le problème du routage, en le réduisant à un choix de spine pour chaque communication entre deux serveurs.

Nous avons ensuite identifié des scénarios de panne entraînant une réduction non uniforme de la bande passante entre les commutateurs. Par exemple, nous démontrons que si deux commutateurs leaf (les commutateurs de la couche inférieure d'un fat-tree) subissent des pannes de liens impliquant différentes spines, la réduction de bande passante entre ces deux commutateurs leaf sera encore plus importante que celle qu'ils subissent avec les autres commutateurs leaf. Ce scénario complique le routage, car il doit également être pris en compte lors de l'ordonnancement pour garantir qu'une solution de routage existe pour chaque scénario de panne sans congestion.

Pour résoudre ce problème, nous proposons un algorithme d'ordonnancement respectant une propriété clé que nous définissons : chaque commutateur leaf ne communique qu'une seule fois par phase avec tous les autres commutateurs leaf. Ainsi, quelle que soit la réduction de bande passante, tant que chaque commutateur leaf reste connecté à tous les autres, une solution de routage sans congestion existe. Notre algorithme d'ordonnancement est également optimal, car il respecte le nombre minimal de phases nécessaires pour réaliser un échange all-to-all avec réduction de bande passante, nombre calculé à l'aide de la formule présentée dans le Chapitre 5.

Pour router ce nouvel algorithme d'ordonnancement, nous proposons un algorithme de routage qui gère les scénarios de panne où la réduction de bande passante est uniforme entre les commutateurs. Pour les cas où cette réduction est non uniforme, nous proposons un modèle de Programmation Linéaire Entières (ILP) afin d'assigner une spine à chaque flux de communication.

Nous évaluons ensuite notre solution de Routage et d'Ordonnancement Optimisés et Adaptatifs aux Pannes (FORS) en termes de débit et la comparons à l'état de l'art. FORS surpasse les solutions existantes. Par exemple, avec une panne de lien dans le réseau, il atteint un débit de 137 Gbps par serveur contre 141 Gbps sans panne, et ce débit diminue progressivement à mesure que le nombre de pannes augmente. Pour rappel, les autres approches de planification atteignent au mieux 95 Gbps. Ainsi, FORS permet d'obtenir un débit global d'environ 44 Tbps avec une panne, tandis que les autres approches atteignent au mieux 30 Tbps. FORS constitue donc une alternative efficace pour gérer les liens instables (liens qui flappent) qui rendent le réseau instable sur de longues périodes, comme démontré au Chapitre 4. Il suffirait de désactiver un lien instable dès qu'il commence à fluctuer et de déployer FORS sur le réseau jusqu'à ce que le lien retrouve sa stabilité.

Nous évaluons également le temps de calcul de FORS en simulant divers scénarios de panne aléatoires, allant d'une panne unique à la perte de la moitié des liens du réseau. Le temps de calcul minimal est d'environ 27 secondes, le maximum est d'environ 37 secondes, et la moyenne est d'environ 30 secondes, rendant notre solution de routage applicable au réseau DAQ étudié.

Enfin, nous discutons de la déployabilité de FORS sur des topologies de réseau plus larges et des contraintes de déploiement liées aux limitations d'Infiniband, que nous espérons voir résolues dans de futures versions.

7.4.2 Perspectives

Une première perspective pour améliorer notre travail serait d'étudier plus en détail la déployabilité de notre solution Fault-Adaptive Optimized Routing and Scheduling (FORS) avec Ethernet. Il serait intéressant d'examiner comment notre algorithme de routage pourrait être implémenté dans un réseau Ethernet. Dans la section 6.7.2, nous avons expliqué dans le Chapitre 6 qu'InfiniBand implémente un routage multipath à l'aide du LID Mask Control (LMC), un masque qui attribue plusieurs Local Identifiers (LIDs) à un serveur de destination. Avec Ethernet, une stratégie comparable pourrait consister à encoder l'épine dorsale (spine) à utiliser comme chemin de routage directement dans l'adresse IP et à appliquer un masque, similaire au LMC, pour déterminer le chemin de routage. Chaque destination recevrait alors autant d'adresses IP qu'il y a de spines dans la topologie fat-tree. FORS serait moins contraint dans un réseau Ethernet, puisqu'il permettrait d'encoder bien plus que la limite de 128 adresses par destination imposée par InfiniBand.

Actuellement, le réseau que nous avons étudié utilise InfiniBand pour ses mécanismes de contrôle de flux et ses cartes réseau InfiniBand à 200 Gbps, une option qui n'était pas disponible avec Ethernet au moment de la conception du réseau. Cependant, Ethernet est plus standardisé et permet l'utilisation d'autres stratégies de routage, telles que Equal-Cost Multi-Path (ECMP). ECMP répartit le trafic sur plusieurs chemins. Ici, comme nos différents flux sont globalement de taille équivalente, ECMP pourrait nous aider à réduire la congestion du réseau.

De manière générale, la déployabilité de FORS devrait être étudiée plus en profondeur sur les réseaux InfiniBand, notamment en développant notre propre moteur de routage pour déployer FORS dynamiquement. Actuellement, FORS ne peut pas être généré automatiquement en cas de défaillance, car nous devons créer un fichier contenant les tables de routage. Ce fichier inclut les routes pour atteindre un serveur de destination à chaque phase et pour chaque commutateur, tout en évitant les ports en panne. Avec l'implémentation actuelle d'OpenSM, si un port en panne est référencé dans la table de routage, OpenSM doit recharger de nouvelles tables de routage à partir de notre fichier de configuration. Pendant ce temps, les paquets sont perdus. Pour remédier à cela, nous devrions développer notre propre moteur de routage capable de calculer notre solution d'ordonnancement et de routage, puis de générer dynamiquement les tables de routage associées. Cela permettrait au contrôleur de basculer dynamiquement vers notre solution en cas de panne, de la même manière que Ftree bascule par défaut vers Min-Hop en cas de panne, sans nécessiter de redémarrer le contrôleur OpenSM.

Une fois FORS déployé sur le réseau DAQ de LHCb, un autre axe de travail futur serait d'évaluer ses performances en termes de fail-over. Plus précisément, il serait intéressant de mesurer le temps écoulé entre l'occurrence d'une panne et le moment où le réseau se stabilise avec la solution FORS. De plus, nous devrions suivre la fréquence et la durée d'utilisation de FORS sur plusieurs mois, puis calculer la quantité de données préservées grâce à FORS. Cela pourrait être réalisé en comparant le débit atteint avec FORS au débit de la solution d'ordonnancement et de routage actuellement déployée, en s'appuyant sur les mesures présentées au Chapitre 4.

Un autre domaine de recherche intéressant serait de définir formellement la complexité du problème de routage lorsque la réduction de bande passante est non uniforme. Pour cela, nous avons proposé un modèle de Programmation Linéaire en Nombres Entières (ILP), comme décrit dans la Section 6.5. Bien que nous ayons tenté de développer un algorithme de routage pour résoudre ce problème, nous n'avons pas réussi. Nous pensons que cette difficulté provient du fait que notre problème pourrait être un Multi-Commodity Flow Problem, plus précisément un Unsplittable Flow Problem (UFP). Dans un UFP, chaque flux de communication entre une source et une destination dans un réseau doit emprunter un seul chemin et ne peut pas être réparti sur plusieurs chemins tout en respectant les contraintes de capacité des liens. Ce problème est connu pour être NP-difficile, et définir formellement notre problème de routage avec réduction de bande passante non uniforme comme un UFP pourrait offrir un aperçu plus approfondi de sa complexité.

La Programmation par Contraintes (CP) est une technique utilisée pour trouver des solutions satisfaisant un ensemble de contraintes, plutôt que d'optimiser une fonction objective comme en programmation linéaire. La CP est particulièrement efficace pour résoudre des problèmes combinatoires où les contraintes sont de nature logique [Google 2024]. Par exemple, elle fonctionne bien lorsque des variables doivent être toutes différentes ou lorsqu'une contrainte impose que si une variable est égale à 1, une autre doit être égale à 0. Cependant, le problème du routage des communications avec une réduction de bande passante non uniforme s'exprime plus naturellement à l'aide de contraintes linéaires, ce qui justifie notre choix de l'ILP dans la Section 6.5. Les contraintes purement logiques ne capturent pas toujours adéquatement la complexité du problème de routage. Bien qu'il soit possible d'exprimer notre problème en utilisant uniquement des contraintes logiques, cela nécessite, à notre connaissance, l'introduction de variables et de contraintes supplémentaires pour reproduire le comportement des contraintes linéaires. Cela complique non seulement le modèle, mais ralentit également le solveur. Nous avons tenté de résoudre le problème de routage avec réduction de bande passante non uniforme en utilisant des contraintes logiques adaptées à la CP. Nous avons employé le solveur CP-SAT open-source de Google OR-Tools[Google 2023], mais nous n'avons pas trouvé de solution dans un délai raisonnable pour la topologie du réseau DAQ étudié. Néanmoins, cette approche pourrait mériter une investigation plus approfondie.

Une autre perspective pour les travaux futurs serait d'adapter notre solution d'ordonnancement à d'autres opérations collectives, comme AllGather. Comme présenté au Chapitre 3, bien qu'AllGather utilise la même bande passante que l'all-to-all, il implique un nombre différent de messages échangés. L'all-to-all nécessite davantage de messages circulant dans le réseau. Notre solution d'ordonnancement pourrait donc probablement être adaptée à ces opérations collectives, qui sont moins complexes à optimiser en cas de panne. De plus, la propriété de notre ordonnancement permettant de gérer une réduction de bande passante non uniforme, ainsi que notre solution de routage, pourraient également s'appliquer à ces opérations.

Enfin, une dernière perspective serait d'étudier comment FORS pourrait être adapté à d'autres topologies, comme HyperX et AB Fat-Tree, qui sont abordées au Chapitre 3. La topologie AB Fat-Tree fait partie de la solution Fault-Tolerant Engineered Network (F10), conçue pour améliorer la tolérance aux pannes. Cette topologie augmente le nombre de chemins entre les leaf switches en cas de panne en brisant la symétrie d'une topologie fat-tree standard. Dans le réseau DAQ de LHCb, la topologie AB fat-tree n'apporte pas d'avantage particulier, car le nombre maximal de chemins entre les leaf switches est déjà atteint grâce à la structure fat-tree à deux couches. Cependant, pour améliorer la tolérance aux pannes dans des topologies fat-tree de plus de deux couches, l'AB Fat-Tree pourrait être une solution intéressante, et FORS pourrait être adapté pour cette architecture. Nous pourrions également étudier l'adaptation de FORS aux topologies HyperX et évaluer si une solution de routage simplifiée serait suffisante pour le routage du trafic, étant donné que la topologie HyperX offre de nombreux chemins redondants.

List of Figures

List of Tables

Bibliography

[Aad *et al.* 2023] G. Aad, B. Abbott, K. Abeling, N.J. Abicht, S.H. Abidi, A. Aboulhorma, H. Abramowicz, H. Abreu and Y. Abulaiti et al. *Fast b-tagging at the high-level trigger of the ATLAS experiment in LHC Run 3.* Journal of Instrumentation, vol. 18, no. 11, page P11006, nov 2023. 15

[Aaij *et al.* 2020] Roel Aaij*et al.* *Allen: A high level trigger on GPUs for LHCb.* Comput. Softw. Big Sci., vol. 4, no. 1, page 7, 2020. 15

[Abdous *et al.* 2021] Sepehr Abdous, Erfan Sharafzadeh and Soudeh Ghorbani. *Burst-tolerant datacenter networks with Vertigo.* In Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies, CoNEXT '21, page 1–15, New York, NY, USA, 2021. Association for Computing Machinery. 38

[Al-Fares *et al.* 2010] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang and Amin Vahdat. *Hedera: dynamic flow scheduling for data center networks.* In Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, NSDI'10, page 19, USA, 2010. USENIX Association. 2, 21, 5

[Amoiridis *et al.* 2023] Vasileios Amoiridis, Thomas Owen James, Dinyar Sebastian Rabady, Dominika Zogatova, Dominique Gigi, Attila Racz, Christian Deldicque, Eric Cano, Emilio Meschi, Philipp Maximilian Brummer*et al.* *The CMS Orbit Builder for the HL-LHC at CERN.* Technical report, CERN, 2023. 14

[Amoiridis, Vassileios *et al.* 2024] 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 and Zogatova, Dominika. *The CMS Orbit Builder for the HL-LHC at CERN.* EPJ Web of Conf., vol. 295, page 02011, 2024. 15

[Avin & Schmid 2021] Chen Avin and Stefan Schmid. Renets: Statically-optimal demand-aware networks, pages 25–39. Society for Industrial and Applied Mathematics, 2021. 26

[Avin *et al.* 2018] Chen Avin, Alexandr Hercules, Andreas Loukas and Stefan Schmid. *rDAN: Toward robust demand-aware network designs.* Information Processing Letters, vol. 133, pages 5–9, 2018. 26

[Bawej *et al.* 2015] Tomasz Bawej, Ulf Behrens, James Branson, Olivier Chaze, Sergio Cittolin, Georgiana-Lavinia Darlea, Christian Deldicque, Marc Dobson, Aymeric Dupont, Samim Erhan, Andrew Forrest, Dominique Gigi, Frank Glege, Guillelmo Gomez-Ceballos, Robert Gomez-Reino, Jeroen Hegeman, Andre Holzner, Lorenzo Masetti, Frans Meijers, Emilio Meschi, Remigius K. Mommsen, Srecko Morovic, Carlos Nunez-Barranco-Fernandez, Vivian O'Dell, Luciano Orsini, Christoph Paus, Andrea Petrucci, Marco Pieri, Attila Racz, Hannes Sakulin, Christoph Schwick, Benjamin Stieger, Konstanty Sumorok, Jan Veverka and Petr Zejdl. *The New CMS DAQ System for Run-2 of the LHC.* IEEE Transactions on Nuclear Science, vol. 62, no. 3, pages 1099–1103, 2015. 1, 11, 4

[Belyaev *et al.* 2017] Nikita Belyaev, Dimitrii Krasnopevtsev, Rostislav Konoplich, Vasily Velikhov and Alexei Klimentov. *High performance computing system in the framework of the Higgs boson studies.* Technical report, ATL-COM-SOFT-2017-089, 2017. 1, 4

[Beni & Cosenza 2022] Majid Salimi Beni and Biagio Cosenza. *An Analysis of Performance Variability on Dragonfly+topology.* In 2022 IEEE International Conference on Cluster Computing (CLUSTER), pages 500–501, Heidelberg, Germany, 2022. IEEE. 25

[Bienkowski *et al.* 2003] Marcin Bienkowski, Miroslaw Korzeniowski and Harald Räcke. *A Practical Algorithm for Constructing Oblivious Routing Schemes.* In Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '03, page 24–33, New York, NY, USA, 2003. Association for Computing Machinery. 23

[Bogdanski *et al.* 2012] Bartosz Bogdanski, Bjørn Dag Johnsen, Sven-Arne Reinemo and Frank Olaf Sem-Jacobsen. *Discovery and Routing of Degraded Fat-Trees.* In 2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies, pages 697–702, Beijing, China, 2012. IEEE. 93

[Bokhari & Parameswaran 2016] Haseeb Bokhari and Sri Parameswaran. Network-on-chip design, pages 1–29. 01 2016. 21

[Borrill *et al.* 2015] Julian Borrill, Reijo Keskitalo and Theodore Kisner. *Big Bang, Big Data, Big Iron: Fifteen Years of Cosmic Microwave Background Data Analysis at NERSC.* Computing in Science & Engineering, vol. 17, no. 3, pages 22–29, 2015. 1, 4

[Bruck *et al.* 1997] J. Bruck, Ching-Tien Ho, S. Kipnis, E. Upfal and D. Weathersby. *Efficient algorithms for all-to-all communications in multiport message-passing systems.* IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 11, pages 1143–1156, 1997. 21

[Cai *et al.* 2021] Zixian Cai, Zhengyang Liu, Saeed Maleki, Madanlal Musuvathi, Todd Mytkowicz, Jacob Nelson and Olli Saarikivi. *Synthesizing optimal collective algorithms.* In Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '21, page 62–75, New York, NY, USA, 2021. Association for Computing Machinery. 11

[CERN 2019] CERN. *Key Facts and Figures – CERN Data Centre*, 2019. [Online]. Available: https://information-technology.web.cern.ch/sites/default/files/CERNDataCentre_KeyInformation_July2019V1.pdf. Accessed on: June 24, 2024. 1, 4

[CERN 2024] CERN. *The Large Hadron Collider*, 2024. [Online]. Available: https://home.cern/science/accelerators/large-hadron-collider. Accessed on: June 24, 2024. 14

[Chan *et al.* 2007] Ernie Chan, Marcel Heimlich, Avi Purkayastha and Robert van de Geijn. *Collective communication: theory, practice, and experience: Research Articles.* vol. 19, no. 13, page 1749–1783, sep 2007. 11

[Chao *et al.* 2012] Hung-Lin Chao, Yean-Ru Chen, Sheng-Ya Tung, Pao-Ann Hsiung and Sao-Jie Chen. *Congestion-aware scheduling for NoC-based reconfigurable systems.* In 2012 Design, Automation Test in Europe Conference Exhibition (DATE), pages 1561–1566, 2012. 21

[Chatti *et al.* 2010] Majed Chatti, Sami Yehia, Claude Timsit and Soraya Zertal. *A hypercube-based NoC routing algorithm for efficient all-to-all communications in embedded image and signal processing applications.* In 2010 International Conference on High Performance Computing Simulation, pages 623–630, 2010. 21

[Chirkov & Wentzlaff 2023] Grigory Chirkov and David Wentzlaff. *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, page 410–422, New York, NY, USA, 2023. Association for Computing Machinery. 2, 5

[Colbourn & Dinitz 2006] C.J. Colbourn and J.H. Dinitz, editors. Handbook of combinatorial designs. Chapman and Hall/CRC, 2nd édition, 2006. 48

[Contini *et al.* 2023] Nicholas Contini, Bharath Ramesh, Kaushik Kandadi Suresh, Tu Tran, Ben Michalowicz, Mustafa Abduljabbar, Hari Subramoni and Dhabaleswar Panda. *Enabling Reconfigurable HPC through MPI-based Inter-FPGA Communication.* In Proceedings of the 37th International Conference on Supercomputing, ICS '23, page 477–487, New York, NY, USA, 2023. Association for Computing Machinery. 2, 5

[Czechowski *et al.* 2012] Kenneth Czechowski, Casey Battaglino, Chris McClanahan, Kartik Iyer, P.-K. Yeung and Richard Vuduc. *On the Communication Complexity of 3D FFTs and Its Implications for Exascale.* In Proceedings of the 26th ACM International Conference on Supercomputing, ICS '12, page 205–214, New York, NY, USA, 2012. Association for Computing Machinery. 11

[Dalcin *et al.* 2019] Lisandro Dalcin, Mikael Mortensen and David E. Keyes. *Fast parallel multidimensional FFT using advanced MPI.* Journal of Parallel and Distributed Computing, vol. 128, pages 137–150, 2019. 11

[Dandapanthula 2011] Nishanth Dandapanthula. Infiniband network analysis and monitoring using opensm. Master's thesis, The Ohio State University, 2011. 34

[de O. Souza *et al.* 2022] Otavio A. de O. Souza, Olga Goussevskaia and Stefan Schmid. *CBNet: Demand-aware tree topologies for Reconfigurable Datacenter Networks.* Computer Networks, vol. 213, page 109090, 2022. 26

[Doi & Negishi 2010] Jun Doi and Yasushi Negishi. *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, pages 1–9, New Orleans, LA, USA, 2010. IEEE. 11, 24

[Domke & Hoefler 2016] Jens Domke and Torsten Hoefler. *Scheduling-Aware Routing for Supercomputers.* In SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pages 142–153, Salt Lake City, UT, USA, 2016. IEEE. 22

[Domke *et al.* 2019a] Jens Domke, Satoshi Matsuoka, Ivan R. Ivanov, Yuki Tsushima, Tomoya Yuki, Akihiro Nomura, Shin'ichi Miura, Nie McDonald, Dennis L. Floyd and Nicolas Dubé. *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, New York, NY, USA, 2019. Association for Computing Machinery. 26

[Domke *et al.* 2019b] Jens Domke, Satoshi Matsuoka, Ivan Radanov, Yuki Tsushima, Tomoya Yuki, Akihiro Nomura, Shin'ichi Miura, Nic McDonald, Dennis Lee Floyd and Nicolas Dubé. *The First Supercomputer with HyperX Topology: A Viable Alternative to Fat-Trees?* In 2019 IEEE Symposium on High-Performance Interconnects (HOTI), pages 1–4, Santa Clara, CA, USA, 2019. IEEE. 26

[Dorelli *et al.* 2022] John Dorelli, Chris Bard, Thomas Chen, Daniel Silva, Luiz Fernando Guedes dos Santos, Jack Ireland, Michael Kirk, Ryan McGranaghan, Ayris Narock, Teresa Nieves-Chinchilla, Marilia Samara, M. Sarantos, Pete Schuck and Barbara Thompson. *Deep Learning for Space Weather Prediction: Bridging the Gap between Heliophysics Data and Theory*, 12 2022. 1, 4

[Drung & Rosenstock 2017] Benjamin Drung and Hal Rosenstock. *Current OpenSM Routing*, 2017. [Online]. Available: `https://github.com/linux-rdma/opensm/blob/master/doc/current-routing.txt`. Accessed on: June 24, 2024. 93

[Elasticsearch 2024] Elasticsearch. *Centralize, transform and stash your data*, 2024. [Online]. Available: `https://www.elastic.co/logstash`. Accessed on: June 24, 2024. 31

[et al. 2024] Hal Rosenstock et al. *opensm(8) - Linux man page*, 2024. [Online]. Available: `https://linux.die.net/man/8/opensm` Accessed on: June 24, 2024. 99

[experts 2023a] Gurobi experts. *Gurobi Optimization Inc. Gurobi optimizer reference manual*, 2023. [Online]. Available: `http://www.gurobi.com`. Accessed on: June 24, 2024. 68, 89

[experts 2023b] Gurobi experts. *max_()*, 2023. Online. Available: `https://www.gurobi.com/documentation/10.0/refman/py_max_.html`. Accessed on: June 24, 2024. 100

[experts 2023c] Gurobi experts. *Start*, 2023. Online. Available: `https://www.gurobi.com/documentation/current/refman/start.html`. Accessed on: June 24, 2024. 91

[Falcón 2015] Raúl M. Falcón. *Enumeration and classification of self-orthogonal partial Latin rectangles by using the polynomial method.* European Journal of Combinatorics, vol. 48, pages 215–223, 2015. Selected Papers of EuroComb'13. 66

[Feng *et al.* 2023] Guangnan Feng, Dezun Dong, Shizhen Zhao and Yutong Lu. *GRAP: Group-Level Resource Allocation Policy for Reconfigurable Dragonfly Network in HPC.* In Proceedings of the 37th International Conference on Supercomputing, ICS '23, page 437–449, New York, NY, USA, 2023. Association for Computing Machinery. 2, 5

[Francois & Bonaventure 2008] Pierre Francois and Olivier Bonaventure. *Avoiding Transient Loops During the Convergence of Link-State Routing Protocols.* Networking, IEEE/ACM Transactions on, vol. 15, pages 1280–1292, 01 2008. 34

[Geoffray & Hoefler 2008] Patrick Geoffray and Torsten Hoefler. *Adaptive Routing Strategies for Modern High Performance Networks.* In 2008 16th IEEE Symposium on High Performance Interconnects, pages 165–172, 2008. 22

[Gill *et al.* 2011] Phillipa Gill, Navendu Jain and Nachiappan Nagappan. *Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications.* SIGCOMM Comput. Commun. Rev., vol. 41, no. 4, page 350–361, aug 2011. 1, 29, 73, 92, 5

[Gomez *et al.* 2007] C. Gomez, F. Gilabert, M.E. Gomez, P. Lopez and J. Duato. *Deterministic versus Adaptive Routing in Fat-Trees.* In 2007 IEEE International Parallel and Distributed Processing Symposium, pages 1–8, 2007. 23

[Google 2023] Google. *OR-Tools - Google Optimization Tools*, 2023. Online. Available: `https://developers.google.com/optimization/cp/cp_solver`. Accessed on: June 24, 2024. 108, 15

[Google 2024] Google. *Constraint Optimization*, 2024. Online. Available: `https://developers.google.com/optimization/cp`. Accessed on: June 24, 2024. 108, 14

[Griner *et al.* 2021] Chen Griner, Johannes Zerwas, Andreas Blenk, Manya Ghobadi, Stefan Schmid and Chen Avin. *Cerberus: The Power of Choices in Datacenter Topology Design - A Throughput Perspective.* Proc. ACM Meas. Anal. Comput. Syst., vol. 5, no. 3, dec 2021. 26

[Hensgen *et al.* 1988] Debra Hensgen, Raphael Finkel and Udi Manber. *Two algorithms for barrier synchronization.* International Journal of Parallel Programming, vol. 17, pages 1–17, 02 1988. 12, 41, 92

[Hoefler *et al.* 2008] Torsten Hoefler, Timo Schneider and Andrew Lumsdaine. *Multistage switches are not crossbars: Effects of static routing in high-performance networks.* In 2008 IEEE International Conference on Cluster Computing, pages 116–125, 2008. 22

[Huang *et al.* 2020] Jheng-Yu Huang, Ming-Hung Hsu and Chung-An Shen. *A Novel Routing Algorithm for the Acceleration of Flow Scheduling in Time-Sensitive Networks.* Sensors (Basel, Switzerland), vol. 20, 2020. 22

[Huang *et al.* 2023] Jiajun Huang, Sheng Di, Xiaodong Yu, Yujia Zhai, Jinyang Liu, Yafan Huang, Ken Raffenetti, Hui Zhou, Kai Zhao, Zizhong Chen*et al.* *gzccl: Compression-accelerated collective communication framework for gpu clusters.* arXiv preprint arXiv:2308.05199, 2023. 2, 5

[Huseyin & İmre 2018] Temuçin Huseyin and Kayhan İmre. *Scheduling Computation and Communication on a Software-Defined Photonic Network-on-Chip*

Architecture for High-Performance Real-Time Systems. Journal of Systems Architecture, vol. 90, 08 2018. 21

[Izzi & Massini 2020] Daniele Izzi and Annalisa Massini. *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), pages 1065–1072, Yanuca Island, Cuvu, Fiji, 2020. IEEE. 2, 21, 5

[Izzi & Massini 2022] Daniele Izzi and Annalisa Massini. *All-to-All Personalized Communication on Fat-Trees Using Latin Squares.* In 2022 International Conference on Software, Telecommunications and Computer Networks (SoftCOM), pages 1–6, Split, Croatia, 2022. IEEE. 2, 13, 5

[Izzi & Massini 2023] Daniele Izzi and Annalisa Massini. *Realizing Optimal All-to-All Personalized Communication Using Butterfly-Based Networks.* IEEE Access, vol. 11, pages 51064–51083, 2023. 2, 21, 5

[Jacobs 2010] Joan Jacobs. *D-Mod-K Routing Providing Non-Blocking Traffic for Shift Permutations on Real Life Fat Trees.* In Computer Science, Engineering, 2010. 16

[Jereczek *et al.* 2015] Grzegorz Jereczek, Giovanna Lehmann Miotto and David Malone. *Analogues between tuning TCP for Data Acquisition and datacenter networks.* In 2015 IEEE International Conference on Communications (ICC), pages 6062–6067, 2015. 1, 11, 4

[Jereczek 2017] Grzegorz Jereczek. *Software switching for high throughput data acquisition networks.* PhD thesis, National University of Ireland, Maynooth (Ireland), 2017. 14

[Karp 1972] Richard M. Karp. Reducibility among combinatorial problems, pages 85–103. Springer US, Boston, MA, 1972. 89

[Karp 1975] Richard M Karp. *On the computational complexity of combinatorial problems.* Networks, vol. 5, no. 1, pages 45–68, 1975. 86, 88

[Kasan *et al.* 2022] Hans Kasan, Gwangsun Kim, Yung Yi and John Kim. *Dynamic Global Adaptive Routing in High-Radix Networks.* In Proceedings of the 49th Annual International Symposium on Computer Architecture, ISCA '22, page 771–783, New York, NY, USA, 2022. Association for Computing Machinery. 22

[Kim *et al.* 2008] John Kim, Wiliam J. Dally, Steve Scott and Dennis Abts. *Technology-Driven, Highly-Scalable Dragonfly Topology.* In 2008 International Symposium on Computer Architecture, pages 77–88, 2008. 24, 25, 18

[Kleinberg 1996] J.M. Kleinberg. *Single-source unsplittable flow.* In Proceedings of 37th Conference on Foundations of Computer Science, pages 68–77, 1996. 88

[Kopeliansky 2023] Revital Kopeliansky. *ATLAS Trigger and Data Acquisition upgrades for the High Luminosity LHC.* Technical report, CERN, Geneva, 2023. 15

[Krawczyk, Rafał Dominik *et al.* 2021] Krawczyk, Rafał Dominik, Pisani, Flavio, Colombo, Tommaso, Frank, Markus and Neufeld, Niko. *Ethernet evaluation in data distribution traffic for the LHCb filtering farm at CERN.* EPJ Web Conf., vol. 251, page 04001, 2021. 98

[Kumar & Kale 2004] S. Kumar and L.V. Kale. *Scaling all-to-all multicast on fat-tree networks.* In Proceedings. Tenth International Conference on Parallel and Distributed Systems, 2004. ICPADS 2004., pages 205–214, 2004. 23

[Lakhotia *et al.* 2021] Kartik Lakhotia, Fabrizio Petrini, Rajgopal Kannan and Viktor Prasanna. *In-network reductions on multi-dimensional HyperX.* In 2021 IEEE Symposium on High-Performance Interconnects (HOTI), pages 1–8, 2021. 26

[Leiserson et al. 1992] Leiserson et al. *The Network Architecture of the Connection Machine CM-5 (Extended Abstract).* SPAA'92, page 272–285. Association for Computing Machinery, 1992. 7

[Leiserson 1985] Charles E. Leiserson. *Fat-trees: Universal networks for hardware-efficient supercomputing.* IEEE Transactions on Computers, vol. C-34, no. 10, pages 892–901, 1985. 9

[Leung *et al.* 2020] Carson K. Leung, Oluwafemi A. Sarumi and Christine Y. Zhang. *Predictive Analytics on Genomic Data with High-Performance Computing.* In 2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pages 2187–2194, Seoul, Korea (South), 2020. IEEE. 1, 4

[LHCb Collaboration 2014] LHCb Collaboration. *LHCb Trigger and Online Upgrade Technical Design Report.* Technical report, CERN, Geneva, 2014. 1, 4

[LHCb Collaboration 2020] LHCb Collaboration. *LHCb Upgrade GPU High Level Trigger Technical Design Report.* Technical report, CERN, Geneva, 2020. 1, 4

[Li & Chu 2019] Yamin Li and Wanming Chu. *Switch Fault Tolerance in a Mirrored K-Ary N-Tree.* In 2019 CITS, pages 1–5, 2019. 9

[Lin *et al.* 2004] Xuan-Yi Lin, Yeh-Ching Chung and Tai-Yi Huang. *A multiple LID routing scheme for fat-tree-based InfiniBand networks.* In 18th International Parallel and Distributed Processing Symposium, 2004. Proceedings., pages 11–, 2004. 95

[List 2024] TOP500 List. *TOP500 List*, 2024. [Online]. Available: https://www.top500.org/lists/top500/2024/06/. Accessed on: June 24, 2024. 7, 15

[Liu *et al.* 2013] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy and Thomas Anderson. *F10: A Fault-Tolerant Engineered Network.* In 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), pages 399–412, Lombard, IL, April 2013. USENIX Association. 26

[Liu *et al.* 2023] Xiaoyan Liu, Zhiwei Xu, Guangwen Liu and Limin Liu. *Intelligent Compound Selection of Anti-cancer Drugs Based on Multi-Objective Optimization.* In 2023 International Conference on Intelligent Supercomputing and BioPharma (ISBP), pages 48–53, Zhuhai, China, 2023. IEEE. 1, 4

[Luppold *et al.* 2018] Arno Luppold, Dominic Oehlert and Heiko Falk. *Evaluating the Performance of Solvers for Integer-Linear Programming.* Technical report, Hamburg University of Technology, Hamburg, Germany, November 2018. 68, 89

[McKay & Wanless 2005] Brendan D McKay and Ian M Wanless. *On the number of Latin squares.* Annals of combinatorics, vol. 9, no. 3, pages 335–344, 2005. 49

[Merindol *et al.* 2018] Pascal Merindol, Pierre David, Jean-Jacques Pansiot, Francois Clad and Stefano Vissicchio. *A Fine-Grained Multi-Source Measurement Platform Correlating Routing Transitions with Packet Losses.* Computer Communications, vol. 129, 08 2018. 31

[MPI 2021] MPI. *MPI(3) man page*, 2021. [Online]. Available: https://www.open-mpi.org/doc/v4.0/man3. Accessed on: June 24, 2024. 11, 40

[MPI 2023] MPI. *MPI: A Message-Passing Interface Standard*, 2023. [Online]. Available: https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf. Accessed on: June 24, 2024. 19

[Munir *et al.* 2016] Ali Munir, Ting He, Ramya Raghavendra, Franck Le and Alex X. Liu. *Network Scheduling Aware Task Placement in Datacenters.* In Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies, CoNEXT '16, page 221–235, New York, NY, USA, 2016. Association for Computing Machinery. 22

[Müller *et al.* 2019] Andreas Müller, Willem Deconinck, Christian Kühnlein, Gianmarco Mengaldo, Michael Lange, Nils Wedi, Peter Bauer, Piotr Smolarkiewicz, Michail Diamantakis, Sarah-Jane Lock, Sami Saarinen, George Mozdzynski, Daniel Thiemert, Michael Glinton, Pierre Bénard, F. Voitus, Charles Colavolpe, Philippe Marguinaud and Nick New. *The ESCAPE project: Energy-efficient Scalable Algorithms for Weather Prediction at Exascale.* Geoscientific Model Development, vol. 12, pages 4425–4441, 10 2019. 11, 14

[Namugwanya *et al.* 2023] Evelyn Namugwanya, Amanda Bienz, Derek Schafer and Anthony Skjellum. *Collective-Optimized FFTs*, 06 2023. 11

[Nvidia 2020] Nvidia. *Collective Operations*, 2020. [Online]. Available: `https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/collectives.html`. Accessed on: June 24, 2024. 2, 19, 5

[Nvidia 2023a] Nvidia. *Appendix – IB Router*, 2023. [Online]. Available: `https://docs.nvidia.com/networking/display/ufmenterpriseumv6112/appendix+%E2%80%93+ib+router`. Accessed on: June 24, 2024. 98

[Nvidia 2023b] Nvidia. *OpenSM*, 2023. [Online]. Available: `https://docs.nvidia.com/networking/display/MLNXOFEDv461000/OpenSM`. Accessed on: June 24, 2024. 16, 17, 23, 36, 39, 49, 56, 93, 95, 98

[OpenSearch 2024] OpenSearch. *Find the truth within your data*, 2024. [Online]. Available: `https://opensearch.org/`. Accessed on: June 24, 2024. 30

[Peng *et al.* 2022] Jintao Peng, Jie Liu, Yi Dai, Min Xie and Chunye Gong. *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), pages 402–409, Melbourne, Australia, 2022. IEEE. 2, 5

[Peng *et al.* 2023] Zhiyong Peng, Jiang Du and Yulong Qiao. *Design of GPU Network-on-Chip for Real-Time Video Super-Resolution Reconstruction.* Micromachines, vol. 14, page 1055, 05 2023. 22

[Petrini & Vanneschi 1997] F. Petrini and M. Vanneschi. *k-ary n-trees: high performance networks for massively parallel architectures.* In Proceedings 11th International Parallel Processing Symposium, pages 87–93, Geneva, Switzerland, 1997. IEEE. 7, 9, 77, 85

[Pippenger 1978] Nicholas Pippenger. *On rearrangeable and non-blocking switching networks.* Journal of Computer and System Sciences, vol. 17, no. 2, pages 145–162, 1978. 7, 80

[Pisani *et al.* 2023a] Flavio Pisani, Tommaso Colombo, Paolo Durante, Markus Frank, Clara Gaspar, Luis Granado Cardoso, Niko Neufeld and Alberto Perro. *Design and Commissioning of the First 32-Tbit/s Event-Builder.* IEEE Transactions on Nuclear Science, vol. 70, no. 6, pages 906–913, 2023. 2, 30, 39, 41, 42, 92, 6

[Pisani *et al.* 2023b] Flavio Pisani, Tommaso Colombo, Paolo Durante, Markus Frank, Clara Gaspar, Luis Granado Cardoso, Niko Neufeld and Alberto Perro. *Design and Commissioning of the First 32-Tbit/s Event-Builder.*

IEEE Transactions on Nuclear Science, vol. 70, no. 6, pages 906–913, 2023. 14

[Prisacari *et al.* 2013a] Bogdan Prisacari, German Rodriguez and Cyriel Minkenberg. *Generalized Hierarchical All-to-All Exchange Patterns.* In 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, pages 537–547, Cambridge, MA, USA, 2013. IEEE. 2, 21, 25, 5

[Prisacari *et al.* 2013b] Bogdan Prisacari, German Rodriguez, Cyriel Minkenberg and Torsten Hoefler. *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, page 139–148, New York, NY, USA, 2013. Association for Computing Machinery. 2, 8, 9, 13, 21, 47, 52, 53, 54, 66, 68, 74, 77, 81, 105, 5, 11

[Prisacari *et al.* 2013c] Bogdan Prisacari, German Rodriguez, Cyriel Minkenberg and Torsten Hoefler. *Fast Pattern-Specific Routing for Fat Tree Networks.* ACM Transactions on Architecture and Code Optimization, vol. 10, pages 1–25, 12 2013. 23, 88, 90, 91

[Räcke 2009] Harald Räcke. *Survey on Oblivious Routing Strategies.* In Klaus Ambos-Spies, Benedikt Löwe and Wolfgang Merkle, editors, Mathematical Theory and Computational Practice, pages 419–429, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. 22, 23

[Rocher-Gonzalez *et al.* 2020] Jose Rocher-Gonzalez, Jesus Escudero-Sahuquillo, Pedro Garcia, Francisco Quiles and Gaspar Mora. *Towards an efficient combination of adaptive routing and queuing schemes in Fat-Tree topologies.* Journal of Parallel and Distributed Computing, vol. 147, 08 2020. 22

[Rocher-González *et al.* 2022] José Rocher-González, Ernst Gunnar Gran, Sven-Arne Reinemo, Tor Skeie, Jesús Escudero-Sahuquillo, Pedro Javier García and Francisco J. Quiles Flor. *Adaptive Routing in InfiniBand Hardware.* In 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pages 463–472, Taormina, Italy, 2022. IEEE. 22

[Rodriguez *et al.* 2009] German Rodriguez, Cyriel Minkenberg, Ramon Beivide, Ronald P. Luijten, Jesus Labarta and Mateo Valero. *Oblivious routing schemes in extended generalized Fat Tree networks.* In 2009 IEEE International Conference on Cluster Computing and Workshops, pages 1–8, 2009. 23

[Roy *et al.* 2015] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter and Alex C. Snoeren. *Inside the Social Network's (Datacenter) Network.* In Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM '15, page 123–137, New York, NY, USA, 2015. Association for Computing Machinery. 11, 38

[Schweissguth *et al.* 2017] Eike Schweissguth, Peter Danielis, Dirk Timmermann, Helge Parzyjegla and Gero Mühl. *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, page 8–17, New York, NY, USA, 2017. Association for Computing Machinery. 88

[Sen *et al.* 2018] Abhijit Sen, Amit Datta and Mallika De. *Fault Tolerant Wormhole Routing for Complete Exchange in Multi-Mesh.* In 2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS), pages 415–420, Belgaum, India, 2018. IEEE. 24

[Sergeev & Del Balso 2018] Alexander Sergeev and Mike Del Balso. *Horovod: fast and easy distributed deep learning in TensorFlow.* arXiv preprint arXiv:1802.05799, 2018. 2, 5

[Shah *et al.* 2023] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi and Rachee Singh. *TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches.* In 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23), pages 593–612, Boston, MA, April 2023. USENIX Association. 11

[Shin & Pinkston 2003] Jeonghee Shin and Timothy Mark Pinkston. *The Performance of Routing Algorithms under Bursty Traffic Loads.* In International Conference on Parallel and Distributed Processing Techniques and Applications, pages 737–743, Las Vegas, Nevada, USA, 2003. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications. 23, 38

[Shpiner *et al.* 2017] Alexander Shpiner, Zachy Haramaty, Saar Eliad, Vladimir Zdornov, Barak Gafni and Eitan Zahavi. *Dragonfly+: Low Cost Topology for Scaling Datacenters.* In 2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB), pages 1–8, Austin, TX, USA, 2017. IEEE. 25

[Singh *et al.* 2021] Rachee Singh, Muqeet Mukhtar, Ashay Krishna, Aniruddha Parkhi, Jitendra Padhye and David Maltz. *Surviving switch failures in cloud datacenters.* SIGCOMM Comput. Commun. Rev., vol. 51, no. 2, page 2–9, may 2021. 2, 29, 30, 42, 92, 5

[Stec 2023] Maciej Stec. *Latin Squares*, 2023. [Online]. Available: `https://www.universityofgalway.ie/media/collegeofscience/schools/schoolofmathematics/files/Latin_Squares.pdf`. Accessed on: July 11, 2024. 48

[Stein *et al.* 2023] Eloise Stein, Quentin Bramas, Tommaso Colombo and Cristel Pelsser. *Fault-adaptive Scheduling for Data Acquisition Networks.* In 2023

IEEE 48th Conference on Local Computer Networks (LCN), pages 1–4, Daytona Beach, FL, USA, 2023. IEEE. 3, 7

[Stein *et al.* 2024] Eloïse Stein, Flavio Pisani, Tommaso Colombo and Cristel Pelsser. *Measuring Performance Under Failures in the LHCb Data Acquisition Network*. IEEE Transactions on Nuclear Science, pages 1–1, 2024. 3, 7

[Subramoni *et al.* 2013] H. Subramoni, D. Bureddy, K. Kandalla, K. Schulz, B. Barth, J. Perkins, M. Arnold and D. K. Panda. *Design of network topology aware scheduling services for large InfiniBand clusters*. In 2013 IEEE International Conference on Cluster Computing (CLUSTER), pages 1–8, Indianapolis, IN, USA, 2013. IEEE. 22

[Subramoni *et al.* 2014] Hari Subramoni, Krishna. Kandalla, Jithin Jose, Karen Tomko, Karl Schulz, Dmitry Pekurovsky and Dhabaleswar K. Panda. *Designing Topology-Aware Communication Schedules for Alltoall Operations in Large InfiniBand Clusters*. In 2014 43rd International Conference on Parallel Processing, pages 231–240, Minneapolis, MN, USA, 2014. IEEE. 22

[Sumanaweera & Liu 2005] Thilaka Sumanaweera and Donald Liu. *Medical image reconstruction with the FFT*. GPU gems, vol. 2, pages 765–784, 2005. 11

[Tariq *et al.* 2021] Umair Tariq, Haider Ali, Lu Liu, John Panneerselvam and James Hardy. *Energy-efficient scheduling of streaming applications in VFI-NoC-HMPSoC based edge devices*. Journal of Ambient Intelligence and Humanized Computing, vol. 12, pages 1–17, 11 2021. 21

[technologies 2020] Mellanox technologies. *QM8700*, 2020. [Online]. Available: `https://network.nvidia.com/files/doc-2020/pb-qm8700.pdf` Accessed on: June 24, 2024. 99

[Update 2011] Product Update. *SIMULIA*, 2011. [Online]. Available: `https://www.fsb.unizg.hr/atlantis/upload/newsboard/22_07_2011__15351_SIMULIA_RSN-May2011.pdf`. Accessed on: June 24, 2024. 1, 4

[Van Lint & Wilson 2001] Jacobus Hendricus Van Lint and Richard Michael Wilson. A course in combinatorics. Cambridge University Press, 2001. 49

[Venkataramani *et al.* 2022] Vanchinathan Venkataramani, Bruno Bodin, Aditi Kulkarni Mohite, Tulika Mitra and Li-Shiuan Peh. *ASCENT: Communication Scheduling for SDF on Bufferless Software-Defined NoC*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 41, no. 10, pages 3266–3275, 2022. 21

[Wang *et al.* 2023a] Ruiqi Wang, Dezun Dong, Fei Lei, Junchao Ma, Ke Wu and Kai Lu. *Roar: A Router Microarchitecture for In-network Allreduce*. In Proceedings of the 37th International Conference on Supercomputing, ICS

'23, page 423–436, New York, NY, USA, 2023. Association for Computing Machinery. 2, 5

[Wang *et al.* 2023b] Weiyang Wang, Manya Ghobadi, Kayvon Shakeri, Ying Zhang and Naader Hasani. *Optimized Network Architectures for Large Language Model Training with Billions of Parameters.* arXiv preprint arXiv:2307.12169, 2023. 2, 5

[Wu *et al.* 2023] Shixun Wu, Yujia Zhai, Jinyang Liu, Jiajun Huang, Zizhe Jian, Bryan Wong and Zizhong Chen. *Anatomy of High-Performance GEMM with Online Fault Tolerance on GPUs.* In Proceedings of the 37th International Conference on Supercomputing, ICS '23, page 360–372, New York, NY, USA, 2023. Association for Computing Machinery. 2, 5

[Yao *et al.* 2014] Fan Yao, Jingxin Wu, Guru Venkataramani and Suresh Subramaniam. *A comparative analysis of data center network architectures.* In 2014 IEEE International Conference on Communications (ICC), pages 3106–3111, Sydney, NSW, Australia, 2014. IEEE. 7

[Yao *et al.* 2022] Yu Yao, Yukun Song, Hu Ge, Ying Huang and Duoli Zhang. *A communication-aware and predictive list scheduling algorithm for network-on-chip based heterogeneous muti-processor system-on-chip.* Microelectronics Journal, vol. 121, page 105367, 2022. 21

[Yazaki *et al.* 2012] Syunji Yazaki, Haruyuki Takaue, Yuichiro Ajima, Toshiyuki Shimizu and Hiroaki Ishihata. *An Efficient All-to-all Communication Algorithm for Mesh/Torus Networks.* In 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications, pages 277–284, Leganes, Spain, 2012. IEEE. 24

[Yigitbasi *et al.* 2010] Nezih Yigitbasi, Matthieu Gallet, Derrick Kondo, Alexandru Iosup and Dick Epema. *Analysis and modeling of time-correlated failures in large-scale distributed systems.* In 2010 11th IEEE/ACM International Conference on Grid Computing, pages 65–72, Brussels, Belgium, 2010. IEEE. 29, 92

[Zahavi *et al.* 2009] Eitan Zahavi, Gregory Johnson, Darren Kerbyson and Michael Lang. *Optimized InfiniBandTM fat-tree routing for shift all-to-all communication patterns.* Concurrency and Computation: Practice and Experience, vol. 22, pages 217 – 231, 11 2009. 2, 13, 16, 22, 23, 24, 36, 47, 48, 52, 73, 77, 93, 5

[Zahavi *et al.* 2014] Eitan Zahavi, Isaac Keslassy and Avinoam Kolodny. *Distributed Adaptive Routing Convergence to Non-Blocking DCN Routing Assignments.* IEEE Journal on Selected Areas in Communications, vol. 32, no. 1, pages 88–101, 2014. 22

[Zahavi 2011] Eitan Zahavi. *Fat-Trees Routing and Node Ordering Providing Contention Free Traffic for MPI Global Collectives.* In 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, pages 761–770, 2011. 23

[Zahid *et al.* 2015] Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdanski, Bjørn Dag Johnsen and Tor Skeie. *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, pages 35–42, Turku, Finland, 2015. IEEE. 24

[Zahid *et al.* 2016] Feroz Zahid, Ernst Gran and Tor Skeie. *Realizing a Self-Adaptive Network Architecture for HPC Clouds*, 2016. 24

[Zerwas *et al.* 2023] Johannes Zerwas, Csaba Györgyi, Andreas Blenk, Stefan Schmid and Chen Avin. *Duo: A High-Throughput Reconfigurable Datacenter Network Using Local Routing and Control.* Proc. ACM Meas. Anal. Comput. Syst., vol. 7, no. 1, mar 2023. 26

[Zhang *et al.* 2017] Qiao Zhang, Vincent Liu, Hongyi Zeng and Arvind Krishnamurthy. *High-resolution measurement of data center microbursts.* In Proceedings of the 2017 Internet Measurement Conference, IMC '17, page 78–85, New York, NY, USA, 2017. Association for Computing Machinery. 38

[Zhao *et al.* 2024a] Liangyu Zhao, Saeed Maleki, Ziyue Yang, Hossein Pourreza, Aashaka Shah, Changho Hwang and Arvind Krishnamurthy. *ForestColl: Efficient Collective Communications on Heterogeneous Network Fabrics.* arXiv preprint arXiv:2402.06787, 2024. 2, 5

[Zhao *et al.* 2024b] Liangyu Zhao, Saeed Maleki, Ziyue Yang, Hossein Pourreza, Aashaka Shah, Changho Hwang and Arvind Krishnamurthy. *ForestColl: Efficient Collective Communications on Heterogeneous Network Fabrics.* ArXiv, vol. abs/2402.06787, 2024. 11, 21

[Zhou *et al.* 2023] Qinghua Zhou, Quentin Anthony, Lang Xu, Aamir Shafi, Mustafa Abduljabbar, Hari Subramoni and Dhabaleswar K. DK Panda. *Accelerating Distributed Deep Learning Training with Compression Assisted Allgather and Reduce-Scatter Communication.* In 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pages 134–144, 2023. 2, 5