Application-Centric Bandwidth Allocation in Datacenters

Mohammadreza Katebzadeh

Doctor of Philosophy

Institute of Computing Systems Architecture
School of Informatics
The University of Edinburgh

2022

Abstract

Today's datacenters host a large number of concurrently executing applications with diverse intra-datacenter latency and bandwidth requirements. Some of these applications, such as data analytics, graph processing, and machine learning training, are data-intensive and require high bandwidth to function properly. However, these bandwidth-hungry applications can often congest the datacenter network, leading to queuing delays that hurt application completion time.

To remove the network as a potential performance bottleneck, datacenter operators have begun deploying high-end HPC-grade networks like InfiniBand. These networks offer fully offloaded network stacks, remote direct memory access (RDMA) capability, and non-discarding links, which allow them to provide both low latency and high bandwidth for a single application. However, it is unclear how well such networks accommodate a mix of latency- and bandwidth-sensitive traffic in a real-world deployment.

In this thesis, we aim to answer the above question. To do so, we develop RPerf, a latency measurement tool for RDMA-based networks that can precisely measure the InfiniBand switch latency without hardware support. Using RPerf, we benchmark a rack-scale InfiniBand cluster in both isolated and mixed-traffic scenarios. Our key finding is that the evaluated switch can provide either low latency or high bandwidth, but not both simultaneously in a mixed-traffic scenario. We also evaluate several options to improve the latency-bandwidth trade-off and demonstrate that none are ideal. We find that while queue separation is a solution to protect latency-sensitive applications, it fails to properly manage the bandwidth of other applications.

We also aim to resolve the problem with bandwidth management for non-latency-sensitive applications. Previous efforts to address this problem have generally focused on achieving max-min fairness at the flow level. However, we observe that different workloads exhibit varying levels of sensitivity to network bandwidth. For some workloads, even a small reduction in available bandwidth can significantly increase completion time, while for others, completion time is largely insensitive to available network bandwidth. As a result, simply splitting the bandwidth equally among all workloads is sub-optimal for overall application-level performance.

To address this issue, we first propose a robust methodology capable of effectively measuring the sensitivity of applications to bandwidth. We then design Saba, an application-aware bandwidth allocation framework that distributes network bandwidth based on application-level sensitivity. Saba combines ahead-of-time application

profiling to determine bandwidth sensitivity with runtime bandwidth allocation using lightweight software support, with no modifications to network hardware or protocols. Experiments with a 32-server hardware testbed show that Saba can significantly increase overall performance by reducing the job completion time for bandwidth-sensitive jobs.

Lay summary

Online applications have become ubiquitous in modern society, serving a wide range of purposes and functions. From social networking and entertainment to finance and education, these platforms have transformed the way we interact, access information, and go about our daily lives. Whether for personal or professional use, online applications have become an essential tool for many people. There are a variety of online applications that are used for different purposes, and each of these applications has its own unique set of requirements when it comes to the network. For example, some applications are designed to handle large volumes of data and require a network with a high bandwidth in order to function effectively. Other applications may not need as much bandwidth, but instead, require ultra-low latency in order to perform well. In either case, it's important to carefully consider the network requirements of different applications in order to ensure optimal performance.

Online applications are deployed in modern datacenters, where hundreds to thousands of diverse applications are run concurrently. In these environments, the network must be shared among all of these applications, which can present challenges in allocating the necessary resources to each application in order to ensure optimal performance. To address the diverse network requirements of the various applications running in the datacenter, operators have begun to deploy high-end networks with low latency and high bandwidth. Such networks are known to provide both low latency and high bandwidth for a single application; however, there is no consensus on how well these networks accommodate the traffic from a mix of applications.

In a real-world deployment, bandwidth-hungry applications often congest the datacenter network, causing the performance and responsiveness of other applications to suffer. Many existing solutions for addressing network congestion fail to adequately improve the performance of applications. While these works can improve network utilization and achieve network-level fairness across applications, they generally ignore application-level performance.

As a step toward improving the performance of shared network datacenter applications, this thesis analyzes the impact of concurrent applications on each other and proposes a novel network bandwidth distribution system that respects the network demands of applications. First, we introduce a performance measurement tool that is capable of accurately measuring the performance of high-end networks. Using our measurement tool, we observe that high-end networks may struggle to provide both low latency and high bandwidth simultaneously in the presence of diverse applications.

Next, propose a robust methodology capable of effectively measuring the network requirements of applications. Finally, we design an application-aware bandwidth allocation scheme, that uses the knowledge about the network demands of applications to distribute bandwidth among applications with the goal of improving their performance.

Acknowledgements

My journey through the Ph.D. program has been a challenging and rewarding experience that has taught me a great deal about myself and the world around me. It has pushed me to think critically, communicate effectively, and problem-solve creatively. It has also helped me develop new skills, such as time management and organization, which will serve me well in my future endeavors. It could not have been completed without the support and inspiration of the people around me.

First and foremost, I would like to express my heartfelt appreciation to my principal advisor, Boris Grot, for his constant guidance, support, and encouragement. His invaluable insights have been crucial in shaping my research and enabling me to complete this thesis. I am deeply grateful for the countless hours he has dedicated to discussing my work, reading and commenting on drafts, providing feedback and direction, and patiently helping me with my writing skills. His patience, generosity, and belief in me have been a constant source of motivation, and I am immensely grateful for all that he has done for me.

Next, I would like to thank Paolo Costa for his invaluable contributions to my research and for his support and guidance throughout my studies. Paolo has been an invaluable source of expertise, always taking the time to carefully review my work and provide insightful feedback and direction. His expertise and insights have been instrumental in shaping my research and helping me to complete this thesis. I am deeply grateful for his dedication and support.

I would also like to extend my sincere gratitude to Mahesh Marina and Marios Kogias, my thesis examiners. I am truly appreciative of their time and effort in helping mold this thesis into a more polished and insightful piece of research. Their insightful and thought-provoking comments opened up exciting possibilities for future directions in my research.

I am fortunate to have had the opportunity to work with and learn from my teammates in the EASE lab. Amna, Artemiy, Priyank, Antonios, Vasilis, David, and Dmitrii have been invaluable sources of support, always willing to lend a helping hand and offer advice and guidance. Their camaraderie and friendship have made my time in the lab enjoyable and rewarding, and I am deeply grateful for their partnership and friendship.

I would also like to express my gratitude to my dear friends, now scattered around the globe, for their steadfast friendship, assistance, and inspiration. I am forever grateful for the kindness that I received from Arash, Negar, Afshin, Amir, and Mahsa in the past 12 years, and their presence and support have meant the world to me.

I would like to extend my deepest gratitude to my family for their unconditional love and support. My father, Majid, and my mother, Nasrin, have always believed in me and encouraged me to pursue my dreams. Their numerous sacrifices and support have enabled me to complete this journey, and I hope I have made them proud and happy. My sister, Yasna also has been a constant source of support and encouragement; her willingness to cheer me on every step of the way has been an invaluable source of strength and motivation, and I am thankful for her love.

Last but certainly not least, my beloved wife and the love of my life, Zohreh; you deserve special thanks for your unwavering love, support, and sacrifices throughout my Ph.D. journey. Your belief in me, encouragement, and understanding have consistently motivated and sustained me. You have always been there to listen, offer support and advice, and provide a shoulder to lean on during the highs and lows of my studies. Your sacrifices and support have enabled me to focus on my studies and complete this journey. In many ways, you have contributed just as much to my success as I have, and my appreciation and gratitude for you are boundless and could be captured in a never-ending list of thanks. Without your love and support, I would not have been able to complete this journey.

Declaration

I declare that this thesis was composed by myself, and that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Some of the material used in this thesis has been published in the following paper(s):

[66] M.R.S. Katebzadeh, P. Costa, B. Grot. "Saba: Rethinking Datacenter Network Allocation from Application's Perspective", in *Proceedings of the Eighteenth European Conference on Computer Systems (EuroSys)*, 2023.

[64] M.R.S. Katebzadeh, P. Costa, B. Grot. "Evaluation of an InfiniBand Switch: Choose Latency or Bandwidth, but Not Both", in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020.

In addition to the work(s) mentioned above, which form the backbone of this thesis, I also contributed to other relevant publications during my studies including:

[68] A. Katsarakis, V. Gavrielatos, M. R. S. Katebzadeh, A. Joshi, A. Dragojevic, B. Grot, and V. Nagarajan. "Hermes: A fast, fault-tolerant and linearizable replication protocol", in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). ACM*, 2020. *Honorable mention in IEEE Micro Top Picks 2020*.

[130] D. Ustiugov, P. Petrov, M. R. S. Katebzadeh, and B. Grot. "Bankrupt Covert Channel: Turning Network Predictability into Vulnerability", in *Proceedings of the 14th USENIX Workshop on Offensive Technologies (WOOT), co-located with USENIX Security*, 2020.

[65] M.R.S. Katebzadeh, P. Costa, B. Grot. "Smart Priority Assignment in Datacenter Networks", in *the 2nd Young Architect Workshop (YArch)*, 2020.

(*Mohammadreza Katebzadeh)*

To my wife, Zohreh,
my parents, Nasrin and Majid, and my sister, Yasna.

Contents

1 Introduction **1**

1.1 Datacenter application and network trends 2

1.2 Bandwidth allocation in datacenters 4

1.3 Problem discussion 5

1.4 Our Approach 6

1.5 Thesis contributions 9

1.6 Thesis organization 10

2 Background **11**

2.1 Datacenter applications 11

2.1.1 Communication patterns 12

2.1.2 Workloads 14

2.2 Datacenter network architecture 16

2.3 RDMA-based interconnection technologies 18

2.3.1 RDMA verbs 19

2.3.2 RDMA transport 20

2.3.3 RDMA execution path 20

2.4 QoS support in datacenter networks 21

2.4.1 QoS in RDMA networks 22

2.4.2 QoS in non-RDMA Ethernet networks: 22

2.5 Bandwidth allocation in datacenters 23

3 Characterizing the Impact of Bandwidth on Applications **31**

3.1 Methodology 32

3.2 Sensitivity to bandwidth in applications 33

3.3 Does flow-level fairness offer optimal performance? 34

3.4 Why does the bandwidth sensitivity arise? 35

3.5 Implications for future application design . . . 36
3.6 Discussion . . . 37
3.7 Summary . . . 38

4 Saba: Application-Aware Bandwidth Allocation Scheme 41
4.1 Saba overview . . . 43
4.2 Profiler . . . 44
4.2.1 Profiling process . . . 45
4.2.2 Accuracy of sensitivity models . . . 45
4.3 Controller . . . 49
4.3.1 Bandwidth calculation and assignment . . . 50
4.3.2 Bandwidth enforcement . . . 50
4.3.3 Mapping applications to queues . . . 51
4.3.4 Centralized vs distributed controller . . . 55
4.4 Saba library . . . 57
4.4.1 Connection manager . . . 57
4.4.2 Software interface . . . 57
4.5 Implementation . . . 58
4.5.1 Profiler . . . 58
4.5.2 Controller . . . 59
4.5.3 Saba library . . . 59
4.6 Evaluation . . . 60
4.6.1 Methodology . . . 60
4.6.2 Main results . . . 61
4.6.3 Sensitivity studies . . . 62
4.6.4 Simulation results . . . 65
4.6.5 Overhead of the controller . . . 68
4.6.6 Discussion . . . 69
4.7 Summary . . . 72

5 Characterization of an InfiniBand Switch 73
5.1 InfiniBand latency measurement . . . 75
5.2 RPerf . . . 77
5.2.1 Excluding remote-side processing . . . 77
5.2.2 Excluding local-side processing . . . 78
5.2.3 RTT calculation . . . 78

5.3 Evaluation of InfiniBand switches 79
5.3.1 Methodology 80
5.3.2 Performance under one-to-one traffic 81
5.3.3 Coexistence of flows with different types 85
5.4 Attempts to protect latency-sensitive flows 88
5.4.1 Bandwidth-intensive flows with different message sizes 88
5.4.2 Packet scheduling policy at the switch 89
5.4.3 Queue separation through priority levels 93
5.4.4 Discussion 97
5.5 Summary 98

6 Conclusions and Future Work **99**
6.1 Summary of contributions 100
6.2 Limitations and Future Work 101

Bibliography **102**

List of Figures

2.1 Communication patterns in distributed applications. 12
2.2 Execution sequence of RDMA operations. 21

3.1 Impact of available bandwidth on the performance of workloads. . . . 33
3.2 Impact of bandwidth allocation scheme on the performance of two co-running workloads. 34
3.3 Impact of available bandwidth on resource utilization and completion time. 35

4.1 An overview of the main components of Saba. 43
4.2 Details of the offline profiler. 44
4.3 Sensitivity models of SQL and LR workloads with various degrees of polynomial (k). 46
4.4 Impact of degree of the polynomial on the accuracy of sensitivity models. 46
4.5 Impact of dataset size at runtime on the accuracy of sensitivity models. 48
4.6 Impact of the number of nodes at runtime on the accuracy of sensitivity models. 48
4.7 Application-to-PL mapping for the studied Spark workloads ($S = 3$). Groups of workloads are circled. 52
4.8 An example of clustering 8 PLs, assuming that the minimum number of queues supported in the switches is 2. Circles and boxes represent PLs and clusters, respectively. The controller clusters PLs based on the closeness of bandwidth sensitivity of associated applications in a hierarchical manner. 53
4.9 Example of PL-to-queue mapping for two switches at runtime. 55
4.10 High-level overview of a distributed deployment of Saba. 55
4.11 High-level overview of the workflow between the profiler and a distributed controller. 56

4.12 High-level overview of interactions between the software interface, the connection manager, the controller, and a network switch. 58
4.13 Speedup of workloads with Saba over the baseline. 61
4.14 CDF of the average speedup of 500 cluster setups. 61
4.15 Impact of dataset size at runtime on the performance of Saba. 63
4.16 Impact of the number of nodes at runtime on the performance of Saba. 63
4.17 Impact of degree of polynomial on the performance of Saba. 64
4.18 Simulation results. (a) speedup of Saba, ideal max-min, Homa, and Sincronia, all over the baseline. (b) the average speedup with centralized versus distributed controllers. (c) impact of the number of queues on the performance of Saba. 65
4.19 Overhead of a centralized controller. 68

5.1 Ping-pong style RTT calculation. 75
5.2 RTT calculation by RPerf. 79
5.3 Back-to-back setup, where two servers are directly connected. 81
5.4 One-to-one setup, where two servers are connected through a switch. . 81
5.5 RTT calculated by RPerf for different packet sizes with and without the switch. 82
5.6 End-to-end RTT calculated by Perftest and Qperf for different packet sizes with the switch. 83
5.7 Bandwidth for different packet sizes with and without the switch. . . . 84
5.8 Mixed-flow-type setup, where from 1 to 5 servers asynchronously send BI flows (number of servers varies in different runs) and 1 server sends LS flow, all to the same destination. 86
5.9 RTT of LS flow 87
5.10 Total bandwidth of all BI flows 87
5.11 RTT of the LS flow. Note that BI flows have different message sizes in each test. 88
5.12 Total bandwidth achieved by BI flows as a function of the message size. 88
5.13 The impact of the number of BI flows on the RTT of LS flow in the simulator. 90
5.14 Simulation with a multi-hop setup. 92
5.15 RTT of the LS flow in a multi-hop setup. 93
5.16 RTT of the real LS flow in different setups. 94

5.17 Total bandwidth achieved by BI workloads under converged traffic. . . 96

Chapter 1

Introduction

Today's high-performance datacenters have become the backbone of our digital world. The datacenter market size is expected to increase from an estimated 220.0 billion USD at the beginning of 2022 to 343.6 billion USD by 2030 [56]. A key driving force behind the growth of the market is the exponential growth in data volume due to the emergence of mobile and web applications, online storefronts, analytics, social media platforms, and cloud services. Many companies provide such services to upwards of billions of users by increasingly deploying a variety of applications in their datacenters to facilitate several petabytes of data processing every day. For example, reports show that Google offers a range of services, including Google Search, Gmail, YouTube, and Google Maps, to 4.3 billion active users around the world [116]. Reports also show that over 2.87 billion people use at least one of Meta's core products, including Facebook, WhatsApp, Instagram, and Messenger, on a daily basis [117]. Similarly, Amazon provides a mixture of services to over 300 million active users [128]. Offering online services to such a colossal user base is only enabled by the massive computational power of datacenters.

Modern datacenters are immense computing infrastructures, consisting of thousands of multi-core servers. The network that interconnects the servers is the core of the datacenter infrastructure. The sustained growth in demand for datacenter applications continues to stress the datacenter network; for example, due to the need to process ever-growing datasets, network bandwidth demand within datacenters is growing. To remove the network as a potential performance bottleneck, datacenter operators are striving to improve the performance of their datacenter networks by investing in physical network equipment and enhancing network control algorithms. While datacenters have seen rapid innovation and have evolved significantly in terms of both network infrastructure and network control algorithms, ensuring high-performance and reliable

communication in datacenters remains an open problem. One reason for this is that in datacenters, application-level and network-level performance objectives are often misaligned, making it difficult to effectively manage the network and optimize performance. When there is a mismatch between these objectives, it can lead to inefficient use of the network resources, resulting in a decline in the performance of applications.

1.1 Datacenter application and network trends

Datacenter applications: Today's datacenters host a mix of traditional and emerging applications. Many of these applications are distributed and run on multiple servers simultaneously and are constantly exchanging data between these servers; thus, requiring high-performance networking.

Datacenters feature a heterogeneous range of bandwidth-intensive applications, including machine learning training [138, 47, 114, 81, 146, 11], SQL queries [13, 70, 136], graph processing [45, 84, 86] and big-data analytics [102, 115]. To deal with growing data volumes, these workloads embrace parallel frameworks, such as Hadoop, Spark, and TensorFlow [141, 122, 28, 1] that operate in multiple stages: computation and communication stages. During a single computation stage, several computation tasks run in parallel. In these frameworks, intermediate data is transferred over the datacenter network in between successive computation stages, known as the communication stage. To efficiently transfer large volumes of data during this stage, these frameworks often use a bulk communication model with hundreds of connections between servers. Consequently, the bandwidth requirements within the datacenter spike during these communication stages, reflecting the priority of bandwidth-intensive applications.

Meanwhile, some applications, including those relying on disaggregated memory [37, 120, 67, 49, 3, 121] and distributed in-memory storage [36, 83, 82, 87, 19, 32, 63, 92, 39], are latency sensitive and mandate ultra-low network latency to provide the illusion of a scale-up system; thus, in the realm of latency-sensitive applications, a different set of challenges and priorities come to the forefront. Typically, latency-sensitive applications send and receive short messages and make up a minority of bytes sent/received inside a datacenter. In many instances, such as memory disaggregation, achieving the lowest possible per-packet latency (on the order of a few microseconds) is critical to the success of service [37]. In such scenarios, any increase in network latency directly correlates with a decrease in service quality. Furthermore, when the processing

is distributed among multiple servers, it is not sufficient to achieve low average latency since the slowest server determines the actual latency of task completion. For example, consider a hypothetical datacenter comprising multiple servers, each with a typical response time of 10 milliseconds, while the 99th percentile latency for these servers is considerably higher, reaching one second. This means that a non-negligible fraction of requests, representing the tail end of the latency distribution, experience latency exceeding one second. For a fan-out query or when numerous microservices collaborate to produce a result when multiple servers are concurrently engaged to process a user request, the task completion time is dictated by the server with the highest latency, leading to potential delays in the overall response time. Even in scenarios where only a tiny fraction of requests (e.g., one in 10,000) encounters high latency on an individual server, a substantial number of user requests will still endure latency exceeding one second if a sizable number of servers (e.g., 2,000) are involved. For that reason, tail latency (e.g., 99th or 99.9^{th} percentile) is a common metric of interest [27], and minimizing network latency and addressing tail latency become paramount concerns.

Datacenter networks: To address the increasing demands for high bandwidth and low latency in datacenters, datacenter operators have begun deploying high-end networking solutions and introduced fully offloaded network stacks through the use of FPGA-enabled network interface cards (NICs) [20, 150, 34] or natively-offloaded fabrics such as InfiniBand [14, 15]. These networks tend to combine custom fully offloaded network stacks, remote direct memory access (RDMA) capability, and lossless links to provide high end-to-end performance.

In spite of datacenter operators scaling the network bandwidth and using offloaded fabric to meet the demands of hosted applications, the network in datacenters continues bottlenecking the performance of applications [59]. The main reason is that the network in datacenters is shared among coexisting applications, giving rise to possible interference between different applications and resulting in contention. Meanwhile, recent works have shown that most datacenter operators do not build full bisection bandwidth (i.e., non-blocking) networks due to financial concerns and cost limitations. The bandwidth in these datacenters is *oversubscribed*, resulting in applications needing more bandwidth than what is available [57]; thus, congestion is more likely to happen. Consequently, as a result of this persistent oversubscription, congestion continues to be a pervasive and significant problem in datacenters, significantly impeding the performance of critical applications.

1.2 Bandwidth allocation in datacenters

In order to control the network bandwidth contention in the presence of congestion, datacenter networks deploy bandwidth allocation schemes and allocate bandwidth according to an allocation policy. Bandwidth allocation policy defines how to distribute the link capacity among flows when congestion happens. Many bandwidth allocation schemes have been proposed in recent years to optimize bandwidth allocation within datacenters. In this pursuit, datacenter networks often implement traffic classification and Quality of Service (QoS) mechanisms. Traffic classes are used to categorize data traffic based on specific attributes, allowing for differentiated treatment of various types of traffic within the network. QoS encompasses a set of policies and mechanisms that ensure a certain level of service quality for different traffic classes, prioritizing and guaranteeing bandwidth, latency, and other performance parameters according to the needs of each class. These mechanisms collectively enhance the efficiency and effectiveness of bandwidth allocation strategies, enabling datacenters to better meet the diverse demands of their applications and workloads. Most of the bandwidth allocation scheme proposals apply one of the following categories of policies:

- Static reservations address bandwidth guarantees for competing applications. These solutions offer strong protection by granting each application independent bandwidth, but they suffer from two key issues. First, network underutilization occurs when applications do not fully use their reserved bandwidth, leading to inefficient resource usage. Second, static reservations require precise bandwidth demands from each application, a challenging requirement in the dynamic landscape of datacenters.

- Minimum bandwidth guarantees, a more flexible approach than static reservations, establish a minimum absolute bandwidth allocation for each application while striving for work conservation to maximize utilization. This approach allows applications to consume more bandwidth than their guarantees when additional bandwidth is available, promoting better network resource usage. However, similar to static reservations, applications still need to specify their bandwidth requirements, which may not align well with the dynamic nature of datacenter workloads.

- Best-effort sharing, in contrast to the aforementioned approaches, does not necessitate explicit network demand expressions from applications. However, it lacks

deterministic guarantees for network performance among competing applications. The two main categories that follow such an approach are as follows:

- The first category attempts to achieve some variant of max-min fairness at the flow level. Max-min fairness is a widely used allocation policy that provides an optimal isolation guarantee by maximizing the minimum bandwidth allocated to each flow.
- The second category aims to achieve optimal average completion time at the flow level by prioritizing short flows.

Best-effort-sharing solutions are particularly practical for the dynamic and ever-evolving nature of the datacenter landscape. But they have one thing in common: they are *application-agnostic* and focus on optimizing bandwidth allocation at the network level. These application-agnostic approaches can be beneficial in terms of simplicity and practicality, as they do not require a detailed understanding of the specific requirements of each application. These approaches, however, have a key drawback: they do not take into account the actual usage of the bandwidth by the applications. As a result, applications that require more bandwidth than they are allocated may not be able to access the necessary bandwidth, leading to poor performance. In addition, applications that are allocated more bandwidth than they need may be wasting resources, leading to inefficient use of bandwidth.

1.3 Problem discussion

In this section, we delve into two critical problems in modern datacenter networks, where the coexistence of bandwidth-intensive and latency-sensitive applications poses significant challenges. These problems highlight the inadequacy of application-agnostic approaches and the complexities of accurately measuring latency-sensitive application performance in this dynamic environment.

1. Insufficiency of application-agnostic bandwidth allocation schemes: This dissertation argues against the effectiveness of application-agnostic approaches at the application level. It emphasizes that network-level properties, such as flow size, do not provide a comprehensive understanding of application-level networking demands. One fundamental issue with these approaches is their equal treatment of all flows, disregarding their collective impact on application performance. We show that different bandwidth-intensive applications exhibit different degrees of sensitivity to the amount of

network bandwidth available, and such sensitivity cannot be captured through flow-level approaches. Our analysis of an InfiniBand deployment, a high-end networking solution, also shows that these schemes are suboptimal as they do not effectively distribute bandwidth.

2. Precise latency measurement challenges: Assessing the impact of the coexistence of bandwidth-intensive and latency-sensitive applications is imperative in modern datacenters. Nevertheless, in the context of emerging networking technologies, measuring latency presents a formidable challenge. The crux of the issue lies in the fact that while tools for measuring latency are available for traditional network stacks such as TCP/IP, the task becomes considerably more challenging when dealing with RDMA networks, which have become increasingly prevalent in modern datacenters. Specifically, existing tools and methodologies are not tailor-made to accurately measure latency for RDMA-based networks with sub-10-microsecond round-trip time.

1.4 Our Approach

This thesis begins by focusing on bandwidth-intensive applications and subsequently evaluates the performance of latency-sensitive ones. By scrutinizing the behavior of bandwidth-intensive applications, we aim to gain insights into their resource demands and their potential impact on other applications sharing the network infrastructure. Furthermore, we seek to provide a comprehensive understanding of how the intricate interplay between the two types of applications (latency-sensitive and bandwidth-intensive applications) can influence the overall efficiency and reliability of datacenter operations.

1. A case for application-centric bandwidth allocation in datacenters: We first attempt to address the deficiency of existing bandwidth allocation schemes from a new angle that moves higher up in the stack and argues that the solution lies in exposing the performance of applications to the network. We posit that in order to contain the impact of network contention on the performance of applications, the bandwidth allocation scheme must evolve to take the performance of applications into account while distributing the bandwidth.

Goal: Our goal is to design a bandwidth allocation scheme that improves the end-to-end performance of applications. Crucially, we want to ensure that such an allocation scheme is practical and well-suited for the datacenter environments that host diverse applications. In the course of this dissertation, we consider packet latency for latency-sensitive

applications and application completion time for bandwidth-intensive applications as our metrics of interest.

Insight: In the context of optimizing best-effort bandwidth allocation schemes within datacenters, our analysis reveals a crucial insight: *the impact of available network bandwidth on different applications can exhibit significant variations*. Recognizing this variability and the need to develop a more tailored approach to address it, we propose a novel metric called **bandwidth sensitivity**. This metric represents the cornerstone of our thesis and is designed to capture the specific impact of network bandwidth on the completion time of individual applications. By quantifying this relationship, we aim to address the issue with best-effort-sharing policies and provide a more precise and granular understanding of how network bandwidth allocation affects the performance of different applications within the datacenter environment. This innovative contribution seeks to enhance the effectiveness of best-effort bandwidth allocation schemes by tailoring them to the unique characteristics and requirements of each application, ultimately optimizing the overall performance and resource utilization of datacenter networks.

Approach: Instead of aiming for fairness or completion time at the flow level, we want a scheme that takes into account how bandwidth affects each application, such that more bandwidth can be given to applications that are more affected by bandwidth constraints. Such a scheme can learn the bandwidth sensitivity of applications by ahead-of-time profiling and use the sensitivity information to derive the bandwidth share of applications with the goal of maximizing their end-to-end performance.

Challenges: As part of our efforts to develop an application-aware bandwidth allocation scheme, we need to address the following challenges:

- Sensitivity Differentiation: Such a solution requires a robust approach to capture the application's sensitivity to network bandwidth.
- Dynamism: At the datacenter scale, a multitude of applications share the network, with new applications arriving and others terminating or migrating over time. A bandwidth allocation mechanism must be able to handle such dynamism in a timely and resource-effective manner.
- Practicality: To facilitate adoption and maximize generality, a bandwidth allocation scheme should not require changes to deployed hardware and/or network protocols.

Design: We design a novel bandwidth allocation scheme called **Saba**, that leverages the bandwidth sensitivity metric by learning the sensitivity of applications with ahead-of-time profiling and sharing bandwidth among different applications accordingly. We put particular effort into the design of Saba with the goal of being widely deployable in existing datacenters by minimizing resources needed during profiling and the number of queues used in switches. Saba is fully compatible with existing switches, NICs, and switches without requiring any changes to congestion-control protocols and requires only a lightweight shim layer at end hosts.

Summary of results: Experiments with a 32-server hardware testbed show that Saba can significantly increase overall performance across a broad set of workloads by reducing job completion time up to 3.94× for bandwidth-sensitive jobs, while only marginally affecting completion times for the others. Furthermore, we evaluate Saba at scale in simulation with 1,944 servers using synthetic workloads and compare it against InfiniBand, an ideal implementation of max-min fairness, and Homa (the state-of-the-art networking protocol designed for datacenters). Our evaluation shows that Saba improves the average performance of studied workloads by 1.27×, 1.11×, and 1.13× compared to InfiniBand, ideal implementation of max-min fairness, and Homa, respectively.

2. Precise latency measurement for RDMA networks in datacenters: Next, this thesis attempts to evaluate the impact of bandwidth-intensive applications on latency-sensitive ones. However, measuring latency with a high level of precision presents a new challenge in datacenters. Specifically, existing tools are not designed to accurately measure latency in emerging technologies, particularly RDMA-based networks, which are increasingly common in datacenters.

Goal: Our goal is to design a latency measurement tool capable of assessing port-to-port latency in RDMA-based networks, while ensuring it minimizes software stack overhead and remains compatible with commodity gear lacking hardware-based timestamping.

Design: To address this gap, we introduce **RPerf**, a tool crafted to measure latency with high precision. RPerf effectively overcomes the limitations of existing tools, enabling precise latency measurement without the need for costly hardware-based solutions or support for hardware timestamping on NICs. RPerf achieves this by leveraging the primitive operations in RDMA, thus effectively excluding software overheads from latency measurements.

Summary of results: Through the utilization of RPerf, we evaluate our hardware testbed and show that our setup achieves remarkably low latency in an unloaded network,

validating prior research findings. However, our observations also reveal that our RDMA-based switch, even after a variety of attempts to optimize the network at the flow level, struggles to maintain low latency for latency-sensitive flows in the presence of bandwidth-intensive flows, shedding light on a critical performance challenge in datacenter environments.

1.5 Thesis contributions

Thesis Statement

Exposing application-level performance to datacenter networks enables mitigating the impact of the network on co-running applications.

To support our thesis statement, in this dissertation, we make five principal contributions:

1. We provide **a characterization of co-running applications** and find that enforcing network-level max-min fairness or shortest-flow first on a per-flow basis can lead to poor overall application performance when multiple applications are sharing the network. Our research indicates that these approaches are unable to accurately identify the actual bandwidth requirements of individual applications. As a result, they are unable to effectively allocate bandwidth and improve the overall performance of applications.

2. We propose the concept of **bandwidth sensitivity** as a guiding principle for allocating bandwidth among applications and show how this metric can be learned through profiling. By taking into account the relative importance of bandwidth to different applications, we show that it is possible to make more informed decisions about how to allocate bandwidth in order to maximize overall application performance.

3. We present **Saba, an application-aware bandwidth allocation scheme** that utilizes bandwidth sensitivity to make allocation decisions. We compare Saba to a baseline approach using InfiniBand's congestion control and demonstrate that Saba is able to significantly reduce the completion time for bandwidth-sensitive jobs by up to 3.94×, while only causing a minor slowdown (1-5%) for a few bandwidth-insensitive jobs. Overall, Saba is able to improve the average completion time by 1.88×.

4. We introduce **RPerf, a micro-benchmarking tool** that is able to provide sub-microsecond precision measurement of latency. We discuss the limitations of current latency measurement tools and demonstrate how RPerf compares favorably to these existing tools in terms of accuracy and precision.

5. We present a detailed **characterization of the latency and bandwidth performance of an InfiniBand switch**. Our studies reveal that while the switch can provide low per-packet latency (on the order of microseconds) and high bandwidth for a single application, it is unable to offer low latency for a latency-sensitive flow in the presence of bandwidth-intensive flows. In order to improve the performance of coexisting applications, we investigate various strategies such as using different packet sizes and priority levels. However, we find that all of the evaluated approaches are deficient in some respect, have some shortcomings, and do not fully address the issue of balancing latency and bandwidth for multiple applications.

1.6 Thesis organization

The remainder of this thesis is organized as follows:

- **Chapter 2** gives an overview of the background related to the work presented in this thesis.
- **Chapter 3** characterizes the performance of applications in a shared environment. We present the motivation and discuss the shortcomings of conventional, application-agnostic bandwidth allocation schemes.
- **Chapter 4** arrives at the complete design of an application-aware bandwidth allocation scheme, Saba. We compare the Saba against the state-of-the-art bandwidth allocation schemes and demonstrate how Saba outperforms other application-agnostic/aware approaches.
- **Chapter 5** characterizes an InfiniBand switch in a rack-scale deployment. We first introduce RPerf, a precise latency measurement tool. We then use RPerf to conduct a set of experiments and evaluate the switch.
- **Chapter 6** concludes the dissertation, summarizing the key contributions and outlining the directions for future work.

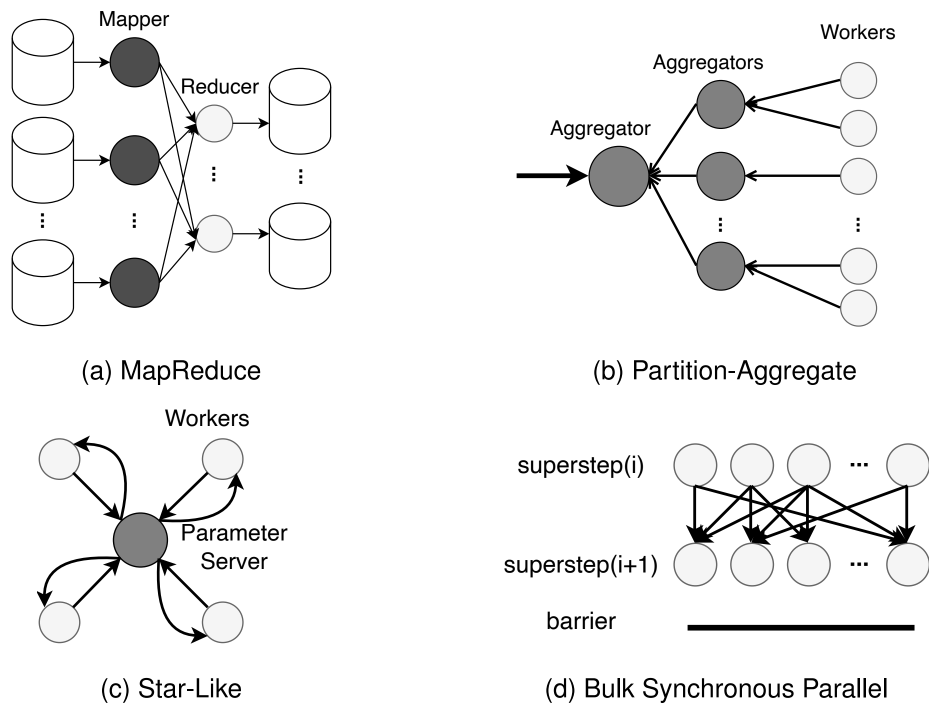
Chapter 2

Background

In this chapter, we present the background relevant to the dissertation. Firstly, we explore the different types of datacenter applications, their communication patterns, and their network demands. Secondly, we describe the architectures of modern datacenters that are used to provide the necessary scalability and performance to meet the demands of today's datacenter applications. Thirdly, we provide essential background information on RDMA technology, InfiniBand, and RoCE networking solutions, which are mentioned throughout this dissertation. Finally, we explore the various types of bandwidth allocation schemes and discuss two flow-level bandwidth allocation schemes and how they aim to ensure efficient use of bandwidth.

2.1 Datacenter applications

Datacenters are facilities that house a large number of servers, storage systems, and networking equipment, all of which are used to support the operation of applications and services. These applications and services may include web servers, email servers, database servers, file servers, and other types of servers. Thus, datacenter applications play a key role in the datacenter ecosystem. Due to ever-growing datasets, datacenter applications are designed to be distributed and run across a cluster of servers. Many of these applications are implemented on top of distributed frameworks to perform data processing (e.g., MapReduce [28] and Spark [141]), graph processing (e.g., Giraph [43]), stream processing (e.g., Storm [129]), machine learning training (e.g., Tensorflow [1]), distributed key-value stores (e.g., Memcached [36]) and distributed database systems (e.g., Spanner [26]). The communication pattern of these frameworks along with the type of workloads running on them determine the networking requirements of the



(a) MapReduce

(b) Partition-Aggregate

(c) Star-Like

(d) Bulk Synchronous Parallel

Figure 2.1: Communication patterns in distributed applications.

applications.

2.1.1 Communication patterns

In datacenters, a communication pattern refers to the specific manner in which data and information flow between various components or nodes within the network. It describes the regular or recurring pathways and interactions that data takes as it travels from its source to its destination, often influenced by the architecture, topology, and design of the datacenter network. These patterns play a crucial role in the design and operation of datacenters, as they determine how data is transmitted, processed, and stored within the datacenter.

There are several basic communication patterns that are commonly used in datacenter applications, including point-to-point communication, publish-subscribe communication, and request-response communication. In addition to these basic communication patterns, there are also more complex patterns that are widely being used in datacenter applications, such as MapReduce, partition-aggregate, star-like, and bulk-synchronous parallel.

MapReduce (Figure 2.1a) is a well-known and widely used programming model in distributed frameworks. In this model, a mapper reads its input from a distributed

filesystem, performs user-defined computations, and then stores intermediate data on the distributed filesystem. A reducer then retrieves the intermediate data from multiple mappers, merges it, and stores the final output on the distributed filesystem. The communication stage of MapReduce, in which the intermediate data is transferred from each mapper to each reducer, is called the *shuffle*. The MapReduce model produces a total of X $\times$ Y flows in the shuffle phase if there are X mappers and Y reducers, as well as at least Y flows for writing the final results.

Partition-aggregate (Figure 2.1b) covers a broad category of data-intensive frameworks, including interactive online services (e.g., Facebook home feeds and Google Search results), big data analytics, graph processing, machine learning training, and real-time stream processing. In this model, a request from a user is passed to multiple workers, and a set of aggregators gather responses from the workers. The aggregation tree can have multiple levels, with the leaf nodes being the workers and the root node being the final aggregator. The worker responses are aggregated and sent back to the user within strict deadlines. If a response cannot be delivered within the deadline, it may be left behind or sent later asynchronously. This model imposes a many-to-few traffic pattern on the datacenter network.

Star-like (Figure 2.1c) is a common communication pattern among traditional machine-learning applications. A machine-learning application that fits a model to data requires multiple iterations until the model's parameters are converged. Because of the large amount of input data that needs to be processed, machine learning frameworks use a distributed architecture. A typical machine-learning framework usually has server nodes, called *parameter servers*, that store globally shared parameters, and worker nodes that perform local computations on their assigned data. In this approach, each worker node can read and update all model parameters. The synchronization required between worker nodes and server nodes to update the model makes network performance a critical factor in the performance of machine learning applications. For example, when the computation is synchronous, the parameter server needs to aggregate the parameters from the workers after each iteration. If one of the workers is slow to respond, the overall training time increases significantly due to the wait for that worker's parameter updates. In modern machine-learning applications, an alternative to the traditional parameter server model is the *All-Reduce* approach. This technique streamlines the synchronization of model parameters across multiple trainers, which involves two key steps: i) Reduce-Scatter, where data from multiple workers is aggregated using a specified reduction function (e.g., summation or minimum), ensuring that each worker retains a portion of

the final data; and ii) All-Gather, where after Reduce-Scatter, the reduced state is shared across all workers, ensuring that each worker possesses the complete and updated model parameters. This technique enforces an all-to-all communication pattern, which can be costly in machine-learning training. Emerging approaches are now tackling the expense associated with this pattern in the All-Reduce technique. These methods optimize inter-worker communication by focusing on data exchanges solely between consecutive layers assigned to different workers, reducing data volume and streamlining exchanges to point-to-point communication. This optimization proves especially advantageous for large-scale models during distributed training [97].

Bulk-synchronous parallel (Figure 2.1d) is a common communication model in distributed computing, used in various frameworks for graph processing, matrix computation, and network algorithms. A bulk-synchronous parallel computation, called a *superstep*, consists of concurrent computation, communication between worker nodes, and barrier synchronization. In this model, the communication phase can be globally optimized for the superstep using explicit barriers at the end of each superstep.

Overall, communication patterns play a critical role in the design and operation of the datacenter. While datacenter applications differ in execution mechanisms and communication patterns, their common trait is that they run across a large number of servers and their logic is organized in multiple computation and communication stages. By understanding the various communication patterns available, it is possible to optimize the performance and efficiency of the network and ensure that it is able to meet the needs of applications.

2.1.2 Workloads

Datacenters accommodate a variety of workloads to provide services to users. These workloads generally fall into one of the following categories:

Latency-sensitive workloads require extremely low network latency and are sensitive to individual message latency. These applications typically send and receive short messages and make up a small percentage of bytes sent and received within a datacenter. An example of this type of workload is *Memcached*, a distributed in-memory key-value store [36]. In a system using Memcached, clients can access data stored on a Memcached server remotely over the network through various operations, including insertion and retrieval. Previous research has shown that even a minor increase of $20\mu s$ in network latency can result in a 25% drop in performance for a Memcached

Table 2.1: Workloads

Workload	Description
Logistic Regression	is a popular machine learning algorithm to predict a categorical response.
Random Forest	is a machine learning algorithm that is widely used for classification and regression tasks.
Gradient Boosted Trees	is a machine learning technique that uses ensembles of decision trees to predict a continuous or categorical outcome.
Support Vector Machines	is a widely used method for large-scale classification tasks.
Nutch Indexing	tests the indexing sub-system in Nutch, a popular open-source (Apache project) search engine.
NWeight	computes associations between two vertices that are n-hop away.
PageRank	measures the importance of each vertex in a graph.
SQL	contains Hive queries (Aggregation and Join) performing the typical OLAP queries.
WordCount	counts the occurrence of each word in the input data.
TeraSort	sorts data as fast as possible to benchmark the performance of the MapReduce/Spark framework.

workload [110], and this drop can be even more significant in the presence of congestion in datacenters.

Bandwidth-intensive workloads such as big-data analytics on top of Hadoop or Spark, distributed machine-learning training, data backup, and VM migrations, employ a bulk communication model that requires exchanging large amounts of data among the nodes, necessitating high bandwidth. *Logistic Regression* is an example of an iterative bandwidth-intensive workload. In Logistic Regression, each iteration includes a large broadcast and a shuffle operation, and it usually takes the workload hundreds of iterations to converge. In each iteration, the processing is held up by the slowest server to complete. Recent work shows that in an implementation of Logistic Regression on Spark, 42% of the job completion time is spent on communication [22]. Such a large fraction of the completion time spent on communication shows the importance of

network bandwidth on bandwidth-hungry workloads such as Logistic Regression.

Throughout this dissertation, we use a range of workloads listed in Table 2.1 from Intel's industry benchmarks running on top of Spark and Flink. These workloads use different combinations of operations, such as map, reduce, filter, and collect in their implementations, resulting in different communication patterns.

2.2 Datacenter network architecture

One of the key challenges in designing and operating datacenters is to ensure that they are able to meet the networking demands of the applications and services they support. This involves not only ensuring that there is sufficient bandwidth and capacity to handle the traffic generated by these applications, but also ensuring that the network is reliable, secure, and able to support the performance and availability requirements of the applications. To address these factors, the architecture of the datacenter network has recently received significant research interest from academia and industry.

A datacenter network architecture is the physical and logical layout of the networking infrastructure within a datacenter and is designed to support the efficient flow of data and communications between servers, storage devices, and other networking components. It is imperative that the datacenter network is well designed so that both the deployment and maintenance of the infrastructure remain cost-effective. This section discusses the various components of a datacenter network architecture, the different types of architectures that are commonly used, and the considerations that go into designing and implementing a datacenter network.

Switch: One of the key components of a datacenter network architecture is the network switch. Switches are used to connect servers and other devices within the datacenter and enable them to communicate with each other. They can be either fixed-configuration switches, which are pre-configured with a fixed number of ports, or programmable switches, which can be expanded with additional modules as needed. In addition to switches, a datacenter network architecture may also include other networking devices such as routers, firewalls, load balancers, and network-attached storage (NAS) devices. These devices perform specialized functions within the network and help to ensure the security and performance of the overall system.

Network topology: Another main component of a datacenter network architecture is the topology of the network that interconnects servers and switches. A datacenter network topology describes the layout of the cabling infrastructure and the way thousands

of datacenter servers are connected via switches. Network topology has a significant impact on the reconfigurability of the datacenter infrastructure to respond to changing application demands and service requirements; thus, a topology must strike a balance between reliability, performance, scalability, and cost around the capabilities and constraints of existing technologies. The most widely-deployed topologies in datacenters are as follows:

Fat-tree is a non-blocking Clos-based topology [124]. It consists of three layers: core, aggregation, and edge.The core layer consists of switches and routers that are interconnected by high-speed links known as backbone connections. The aggregation layer aggregates uplinks from the edge layer to the core layer using higher bandwidth links. The edge layer, which consists of Top of Rack (ToR) switches, connects servers to the datacenter network. The fat-tree topology aims to alleviate bandwidth bottlenecks closer to the core by adding additional links. In this topology, the number of links connecting a switch to its lower layer switches is equal to the number of links connecting the switch to its parent switch.

VL2 [48] is another Clos-based 3-layer network topology that consists of intermediate, aggregation, and ToR switches.In this topology, each ToR switch connects to different aggregation switches, and each aggregation switch is connected to every intermediate switch. The main difference between the fat-tree and VL2 topologies is that in the VL2 topology, switch-to-switch links have much higher capacity than server-to-switch links, which reduces the number of cables required to connect the aggregation and intermediate (core) switches.

Leaf-spine is a 2- or 3-layer network topology composed of spine and leaf layers.In this topology, servers are connected to the leaf layer and every spine switch is directly connected to all leaf switches. The connections between the servers and leaf layer may have a different capacity from the ones connecting leaf switches to the spine. The leaf-spine topology minimizes point-to-point latency, as each packet only has to travel to a spine switch and another leaf switch to reach its destination.

Bisection bandwidth: Network topologies are characterized by their bisection bandwidth, a key factor that reflects the true bandwidth available in a given datacenter network. Bisection bandwidth is defined as the maximum amount of bandwidth in the datacenter measured by bisecting the network at any given point. Non-blocking networks offer full bisection bandwidth, allowing any input port to transfer data to any unused output port at the full line rate. While providing full bisection bandwidth mitigates the network bottlenecks, datacenter operators often deploy *oversubscribed*

networks to contain costs. For example, Facebook's leaf-spine style datacenter network has a 4:1 oversubscription ratio from rack to rack, as described in [112].

Datacenter network architecture considerations: There are several factors that need to be considered when designing and implementing a datacenter network architecture. One of the most important is performance. A datacenter network must meet the networking demands of the applications and services inside the datacenter. The performance of the network can be critical for certain types of applications, such as those that require low latency or high bandwidth. Applications that are sensitive to network performance may require specialized networking equipment or technologies to ensure that they can operate effectively. Another important factor is scalability. A datacenter network must be able to grow and evolve as the needs of the business change, so it is important to design it in a way that allows for easy expansion. Other considerations include security, reliability, and cost.

In conclusion, a datacenter network architecture is a critical component of any datacenter, providing the infrastructure that enables the efficient flow of data and communications within the facility. There are several different components and types of architectures that can be used to create a datacenter network and a range of factors need to be considered when designing and implementing one.

2.3 RDMA-based interconnection technologies

In order to provide both high bandwidth and low latency, datacenter operators are implementing high-performance interconnection solutions that utilize Remote Direct Memory Access (RDMA) technology. RDMA addresses the issues commonly associated with the traditional TCP stack, such as packet processing and data copy overhead, by operating at the network layer below the transport layer where traditional networking protocols such as TCP operate. This allows RDMA to bypass the operating system's overhead and provide direct access to memory, resulting in low latency and high bandwidth.

One of the main benefits of RDMA is its low latency. Because the data is transferred directly from one server's memory to another, there is no need for the data to be processed or routed through the operating system. This results in much lower latency compared to traditional networking technologies that rely on the CPU and the operating system to process and route data and brings the end-to-end network latency down from milliseconds to microseconds.

RDMA is also very efficient in terms of CPU utilization. As the data is transferred

directly between the memories of servers, there is no need for the CPU to process the data or for the data to pass through the operating system. This means that the CPU can be used for other tasks, rather than being tied up with data transfer.

In an RDMA-based fabric, servers are equipped with RDMA-enabled NICs (RNICs) connected through RDMA switches. InfiniBand and RDMA over Converged Ethernet (RoCE) are well-known RDMA-based networking solutions used in today's datacenters.

InfiniBand is a high-speed interconnect technology that connects servers with remote storage and networking devices. Instead of Ethernet, InfiniBand performs RMDA over InfiniBand adapters and switches, removes network software stack overheads, eliminates context switches, and avoids the need for software execution on the remote CPU. High bandwidth, data integrity, and reliability are other important features that make InfiniBand well-suited for high-end datacenter networks. Further, due to InfiniBand's hop-by-hop credit-based flow control, where a sender cannot send packets in excess of the credit amount advertised by the receive buffer at the other end of the link, the transport layer ensures lossless communication.

RoCE is another networking solution, which enables RDMA on Ethernet networks. RoCE offloads the network stack to RNICs to support efficient RDMA transport services over Ethernet and delivers high-performance networking for latency-critical and bandwidth-intensive applications. Early studies have found that RoCE fabric achieves high end-to-end performance only when the underlying Ethernet network is lossless, leading datacenter operators to use Ethernet's Priority Flow Control (PFC) mechanism to minimize packet loss [93]. Studies have also shown that RoCE has higher latency and lower bandwidth compared to InfiniBand [133].

The performance of an RDMA network is influenced by various factors, including the type of RDMA primitives, the transport type, and the quality-of-service (QoS) configuration. In this section, we provide background information on these factors and explain how they interact with each other.

2.3.1 RDMA verbs

In RDMA terminology, a verb refers to the type of communication operation. There are two types of verbs: *two-sided* (send, receive) and *one-sided* (read, write). Two-sided verbs involve both communication endpoints, while one-sided verbs only involve one endpoint (the source). Two-sided communication requires the remote host to pre-post an RDMA receive, and the local host to post an RDMA send. In contrast, the local host

can use an RDMA write primitive to directly write data to the remote host's memory region, or an RDMA read primitive to fetch data from the remote host's memory region without involving the remote host.

RDMA verbs use the asynchronous I/O model, in which data transfers are non-blocking and allow the application to continue executing before a posted request is completed. Both one-sided and two-sided verbs can use a completion signal (CQE) issued by the RNIC to notify the host when a request is finished. The CQE is added to an application-visible completion queue and the application can receive the notification by polling the queue.

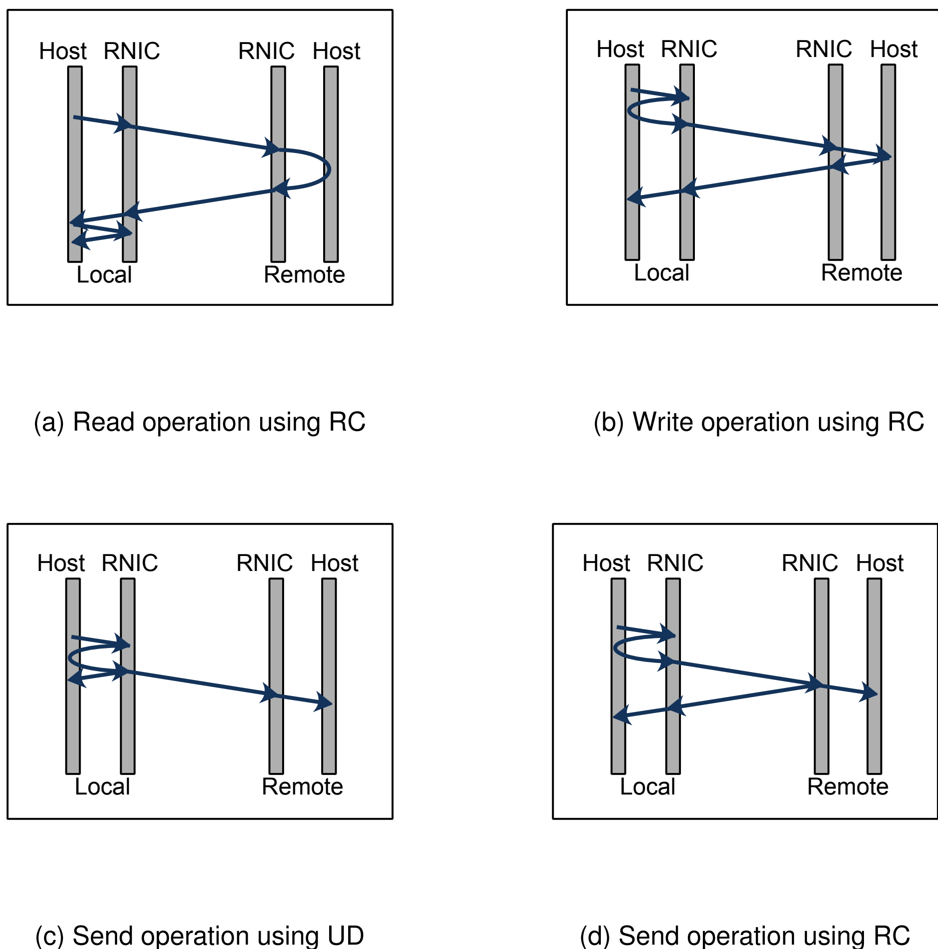
2.3.2 RDMA transport

RDMA offers two transport types: unreliable (UD) and reliable (RC). UD transport only provides two-sided verbs and does not ensure that requests will be delivered. RC transport, in contrast, uses acknowledgments to guarantee the delivery of requests and supports both one-sided and two-sided verbs.

2.3.3 RDMA execution path

The sequence of interactions between communicating hosts in an RDMA transaction depends on the RDMA verb and transport type chosen. Figure 2.2 illustrates the complete sequence of interactions between hosts and RNICs for various RDMA verb and transport pairs.

At the start of each transaction, regardless of the type of verb and transport, the local host posts a request to the local RNIC via an MMIO transaction over PCIe. The local RNIC processes the request based on the verb type specified in the request as follows:
RDMA read: Initially, the local RNIC sends the request over the network fabric. The remote RNIC serves the request through a DMA read from the host's memory hierarchy and sends the data back to the local RNIC. Upon receipt of the data, the local RNIC issues a DMA write to store the data in local memory. The local RNIC then issues another DMA write to add a CQE to the completion queue (Figure 2.2a).
RDMA write: To initiate the transaction, the local RNIC retrieves the payload through a DMA read. The request is then sent over the network fabric, where the remote RNIC stores the data in its host's memory with a DMA write and sends an ACK back to the local RNIC. Upon receiving the ACK, the local RNIC issues a CQE with a DMA write (Figure 2.2b).



(a) Read operation using RC

(b) Write operation using RC

(c) Send operation using UD

(d) Send operation using RC

Figure 2.2: Execution sequence of RDMA operations.

RDMA send: To begin the transaction, the local RNIC retrieves the payload through a DMA read and sends the request over the network fabric. When the remote RNIC receives the request, it sends an ACK (for RC transport) and stores the payload in its host's memory with a DMA write. Depending on the RDMA transport used by the RDMA send request, the local RNIC issues a CQE either immediately after sending the request over the fabric (UD, Figure 2.2c) or after receiving the ACK from the remote RNIC (RC, Figure 2.2d).

2.4 QoS support in datacenter networks

Datacenter networks handle a wide range of traffic patterns and require high QoS to meet the diverse needs of different flows. QoS enables the assignment of priorities

to flows and supplies various network characteristics, such as buffer allocation, flow control, queuing, and scheduling, to each priority.

2.4.1 QoS in RDMA networks

RDMA allows for per-flow differentiation by providing a set of priority levels that can be assigned to flows and exposed to the application developer. These priorities are carried in the header of InfiniBand/RoCE packets as they are transmitted through the network.

InfiniBand uses a set of priority levels, called *Service Levels (SLs)* to hide two of its architectural components that help in achieving QoS:

1) Virtual lane (VL): The concept of VLs allows a physical link to be divided into different logical communication links, each with its own buffering, flow-control, and congestion management resources. A VL arbiter controls the bandwidth usage by selecting flows according to the VL arbitration table. InfiniBand specification specifies that each port must have a minimum of two and a maximum of 16 VLs [14].

2) Virtual lane arbitration: Every network component in the subnet of an InfiniBand fabric has a Service Level to Virtual Lane (SL2VL) mapping table which specifies the VL and priority for each packet. The SL to VL mapping and the priorities set for each VL are configurable in every InfiniBand switch.

In RoCE, the flow differentiation and assigning each flow to a hardware priority level, called *Traffic Class (TC)*, consists of several actions:

1) The application sets the desired priority.

2) This priority is translated into a socket priority.

3) The system administrator maps the socket priority to a user priority.

4) The network/system administrator maps the user priority to a TC.

2.4.2 QoS in non-RDMA Ethernet networks:

In non-RDMA Ethernet networks, QoS employs various techniques to ensure optimal performance for critical applications like VoIP, video conferencing, and online gaming. These techniques include Resource Reservation Protocol (RSVP), queuing, and traffic marking [127].

Traffic Marking: QoS identifies applications that need traffic management and marks their specific traffic. This marking enables routers to create separate virtual queues for

each application. Techniques for marking traffic include Class of Service (CoS) and Differentiated Services Code Point (DSCP).

Class of Service (CoS): CoS is a 3-bit field in Ethernet frames that supports VLANs. It marks traffic at Layer 2 by altering frame header bits, allowing QoS to identify and manipulate traffic based on its class.

Differentiated Service Code Point (DSCP): DSCP assigns different service levels to network traffic by tagging DSCP codes to packets in Layer 3 headers. This enables varying levels of service treatment.

Queuing: Queuing involves creating policies that prioritize specific data streams over others. Routers and switches use high-performance memory buffers called queues for this purpose. A Queue is a high-speed buffer present in a router or switch that stores network traffic until it is ready to be processed or sent in sequence. Routers and switches feature two kinds of hardware queues: ingress (inbound) and egress (outbound) queues. These hardware queues can be classified into two categories: standard queues and strict priority queues. Standard queues treat all traffic equally, without any special treatment or prioritization. In contrast, strict priority queues are specifically reserved for high-priority traffic, making them the preferred choice for QoS-enabled interfaces to handle high-priority packets.

Resource Reservation Protocol (RSVP): RSVP is a transport layer protocol that reserves network resources to achieve different Quality of Service levels for application data streams. RSVP uses path and reservation messages:

Path Messages: Sent from source to receiver, storing path state at each node. These states guide the receiver to reserve network resources on nodes along the path.

Reservation Messages: Sent from receiver to sender, identifying the resources needed for a data stream.

In summary, QoS in non-RDMA Ethernet networks employs techniques like traffic marking (using CoS and DSCP), queuing (with standard and strict priority queues), and RSVP to provide service differentiation and manage network resources effectively.

2.5 Bandwidth allocation in datacenters

Bandwidth allocation is a key aspect of network design and management in datacenters, which refers to the allocation of available bandwidth among different applications within the network according to an allocation policy. Bandwidth allocation policy defines how to distribute the link capacity among flows/applications. There are several ways to apply

a bandwidth allocation policy.

One common approach to bandwidth allocation in datacenters involves the integration of traffic shaping (rate limiting) and congestion control algorithms, where congestion control techniques play a foundational role in this allocation process. Congestion control algorithms are responsible for continuously monitoring the traffic within the network and dynamically adjusting the allocation of bandwidth in response to fluctuations in traffic levels. For instance, when the network experiences congestion, these algorithms may proactively or reactively reduce the allocated bandwidth for specific flows or applications, thereby mitigating overall congestion and maintaining network stability. Such an approach enhances the network performance by ensuring that all applications have equitable access to the necessary bandwidth. Traffic shaping, or rate limiting, is a complementary mechanism that relies on the principles of congestion control and involves setting limits on the bandwidth that different flows or applications can utilize. By doing so, traffic shaping prevents any single flow or application from monopolizing an excessive portion of the available bandwidth.

The general consensus is that the traditional transport-layer protocols, such as TCP, do not perform bandwidth allocation efficiently in datacenters [149, 6, 53, 8]; consequently, network traffic from competing applications interferes with each other, resulting in a severe lack of predictability of application performance. To resolve such a critical issue, a substantial amount of recent research has focused on bandwidth allocation in datacenter networks. These solutions can be categorized into one of the following groups:

1. Static reservations: One area of research focuses on providing bandwidth guarantees to competing applications through static bandwidth reservations [50, 134, 148, 76]. These solutions offer strong protection by ensuring that the bandwidth available to an application is independent of other applications. However, they require applications to explicitly specify their bandwidth requirements. Each application has a virtual network with guaranteed bandwidth based on its demand, and rate limits are statically enforced. Virtual machines are placed appropriately to achieve these guarantees, and an admission control system ensures sufficient bandwidth reservations for all existing applications upon request. While static reservations provide predictable application performance through bandwidth isolation, they are not efficient in utilizing network bandwidth in datacenters. For example, when an application does not fully use its bandwidth reservations, the unused bandwidth cannot be utilized by other applications, leading to wasted bandwidth and inefficient utilization. Additionally, many solutions based

on static reservations are difficult to implement due to their complexity in network management.

2. Minimum guarantees: A less strict version of static reservations is minimum bandwidth guarantees, which provide a minimum absolute bandwidth for each application and aim for *work conservation* to achieve high utilization. With work conservation, applications are permitted to use more bandwidth beyond their guarantees when there is available bandwidth. There are two types of mechanisms that ensure minimum guarantees:

1) End-to-end bandwidth sharing through congestion control, which either assumes that the network core is congestion-free, e.g., Gatekeeper [111] and EyeQ [58], or does not rely on such an assumption, e.g., ElasticSwitch [106] and Hadrian [18]. These works combine admission control and VM placement with congestion control, to ensure minimum bandwidth guarantees and work conservation.

2) In-network bandwidth sharing using weighted fair queuing (WFQ), which offers minimum bandwidth guarantees for tree-based topologies (e.g., fat-tree, VL2, and leaf-spine) by providing WFQ scheduling policy at every switch in the network [105]. However, such an approach lacks practicality, as it requires switches to maintain per-application queues.

3. Network proportionality: In public cloud environments, tenants pay to access resources. Network proportionality is a bandwidth allocation policy that shares bandwidth between tenants in proportion to their payments, similar to how other resources are shared in the cloud. However, implementing this policy can be difficult due to the varying communication patterns of applications, capacity constraints, and the demands of tenants. Additionally, recent research has shown that it is challenging to achieve network proportionality and that there is a trade-off between network proportionality and high utilization [105].

4. Best-effort sharing: Best-effort sharing is another approach to bandwidth allocation and unlike the aforementioned approaches, it does not require applications to explicitly express their network demands. However, it does not provide deterministic guarantees for the network performance of competing applications. Solutions based on best-effort sharing generally apply an allocation policy at the flow level in order to achieve either flow-level fairness, where each flow receives an equal amount of bandwidth compared to other flows, or optimized completion time, where the completion time of flows is proportional to their size [104].

1) Flow-level fairness: Traditionally, per-flow fair scheduling has been advocated

as a way to make Internet bandwidth sharing more efficient and robust. Fairness is achieved when bandwidth is equally allocated among flows and each flow can increase its bandwidth utilization without decreasing the utilization of other flows beyond their allocation. Max-min fairness is a widely-used allocation policy that aims to prevent any flow from monopolizing the network and to ensure that all flows have access to the necessary bandwidth. One of the benefits of max-min fairness is that it can help prevent congestion in the network by ensuring that no flow consumes more than its fair share of available bandwidth. This allows the network to more effectively manage the flow of data and avoid bottlenecks.

Max-min fairness aims at achieving high utilization while maintaining fairness among flows. To this end, max-min fairness maximizes the minimum amount of bandwidth allocated to each flow. In max-min fairness, initially, bandwidth is allocated equally among all flows. If one or more flows cannot utilize their share, the max-min fairness policy allocates the unused bandwidth equally among the rest of the flows until all flows are satisfied or bandwidth is fully allocated. As a result, this allocation policy is work-conserving, meaning that in the presence of sufficient demand, it allocates the whole link capacity. Recently proposed protocols like DCTCP [7], NDP [54], and Swift [71] aim to achieve max-min fairness while keeping low queue utilization. For example, NDP is a receiver-driven flow control mechanism designed to reduce congestion in networks, particularly at the downlink from the ToR to the receivers. When a new message is sent using NDP, the sender transmits the first Bandwidth Delay Product (BDP) of the message at the maximum rate of the NIC. This helps to reduce latency for short messages. After the initial BDP is sent, the sender will not send any more packets for the message until instructed to do so by the receiver. The receiver controls the rate at which new packets are received by sending a 'pull' packet for every data packet it receives. This helps to prevent congestion at the ToR switch by pacing the arrival of packets on the downlink. In this way, the receiver is able to regulate the flow of data to avoid overwhelming the network. NDP requires special switches to operate and modifications to applications.

An ideal implementation of max-min fairness transfers data from flows using a bit-by-bit Round-Robin scheduling algorithm, where one bit of each flow is serviced per round [52]. Such an implementation, however, is not practical as the transmission unit at the link level is a packet not one bit of data. A variety of algorithms have been developed for approximating an ideal max-min fairness implementation by calculating rates and enforcing the share of flows in different settings [42, 30, 119, 143, 103, 75, 44]. Also,

recent studies propose extending max-min fairness to the level of VM pairs and tenants to prevent users from misusing the allocation policy by initiating more connections. NetShare [74] approximates max-min fairness at the tenant level in a hierarchical manner. FairCloud [105] provides weighted max-min fairness on congested links. Seawall [118] addresses VM-level fairness by enforcing VM-to-VM rates for VMs belonging to one tenant. Silo [57] focuses on VM placement and a hypervisor-based packet pacing to provide latency and bandwidth guarantees at the datacenter scale. None of these approaches considers the application-level sensitivity in bandwidth allocation, and hence, they provide sub-optimal performance.

Overall, max-min fairness is an important bandwidth allocation scheme in datacenter networks. By ensuring that all flows receive a fair share of the available bandwidth, the allocation scheme can help to prevent congestion, improve performance, and ensure that all applications receive the bandwidth equally.

2) Optimal flow completion time: One of the assumptions in bandwidth allocation studies is that flow completion time directly translates to application performance [23, 96]. The intuition is that if flows are completed sooner, the communication stages of applications will be completed faster. It is proven that the Shortest-Remaining-Processing-Time (SRPT) scheduling algorithm can achieve optimal average flow completion time among competing applications.

The SRPT algorithm works by continuously monitoring the remaining processing time of each flow and prioritizing the flow with the shortest remaining. This ensures that the flow with the least amount of processing time left is completed as soon as possible, thereby reducing the overall processing time for all flows.

One of the main advantages of SRPT flow scheduling is its simplicity. The algorithm only requires a small amount of data to make its scheduling decisions, making it easy to apply in designing bandwidth allocation schemes. However, SRPT flow scheduling is not without its limitations. One of the main drawbacks is that it can lead to unfair bandwidth allocation, as flows with shorter processing times may be prioritized over those flows with longer processing times. This can lead to long waiting times for flows with longer processing times, which can negatively impact the overall performance of the network. Another drawback of SRPT is that today's datacenter switches do not support the SRPT algorithm, and this scheduling algorithm cannot be ported to the newer scheduling mechanisms like PIFO [125].

Several proposals have borrowed this idea and designed bandwidth allocation schemes and congestion control protocols that implement or approximate the behavior

of the SRPT algorithm, such as pFabric [9], Homa [94], and pHost [38]. Many of these proposals try to overcome the limitations of SRPT by emulating its behavior through using a small number of independent priority-scheduled queues (most of today's commodity switches support eight queues), where all packets from a higher priority queue are sent before packets from lower priority queues. However, the fact that the information about the remaining processing time of flows needed for SRPT is not currently available in today's transport protocols, makes these solutions impractical in the current datacenter settings [152]. pFabric is a transport mechanism that uses priority queues and DCTCP for the rate control mechanism to improve the efficiency and speed of data transfer. It is particularly effective for transferring short messages and is able to achieve low latency while also maximizing the use of network bandwidth. pFabric was the first to discover that using priority queues in this way could significantly reduce the time it takes for data to be transferred or the latency of the message. pFabric, however, requires modifications to the switch hardware in order to support an infinite number of priority levels, which limits its practicality. Homa is another transport protocol that employs a combination of network priority queues and receiver-driven packet scheduling to improve the performance of data transfer in datacenters, particularly for short messages. Like pFabric, it uses priority queues to prioritize short flows, and like NDP, it uses receiver-driven scheduling to regulate the flow of data and prevent congestion at the downlink of hosts. Homa has been shown to be particularly effective at reducing latency, especially under high network loads, and has been found to have lower tail latencies than other transport protocols such as TCP, DCTCP, InfiniBand, and NDP. All of the aforementioned solutions optimize for network-level metrics and ignore the impact of bandwidth sensitivity for different applications, which can result in poor aggregate application performance.

5. Application-aware bandwidth allocation: There have been a few attempts to use the application-level network demands in bandwidth allocation, many of which are specialized for specific types of applications and require software modification. AppSch [78] relies on a priori knowledge of the flow size of all flows in applications and allocates paths for their flows.

In order to bridge the gap between network demands of applications and network-level properties like flow, a new concept called *coflow* [23] (collection of related flows) has been introduced in datacenter networks. A coflow refers to the flow of data between multiple pairs of endpoints that have the same application-level semantics and share a common performance goal. For example, the flows in the shuffle stage between the

mappers and the reducers in MapReduce (Section 2.1.1) can be considered as a coflow. By capturing the collective objective of related flows, coflow enables application-aware bandwidth distribution in datacenter networks. The completion time of coflows has become an important metric in coflow-aware flow scheduling algorithms. These algorithms aim to minimize the coflow completion time while also maximizing the utilization of available bandwidth. There has been significant research focused on optimizing coflow completion time, such as Coflow [23], Orchestra [24], Varys [25], and Sincronia [2]. For example, Sincronia is a network design for coflows that aims to achieve near-optimal performance. It can be implemented on top of any transport layer that supports priority scheduling and uses a greedy mechanism to periodically order unfinished coflows. Each host in the network sets priorities for its flows based on the coflow order, and the scheduling and rate allocation of the flows is then handled by the underlying priority-enabled transport layer. By using this approach, Sincronia is able to achieve high levels of efficiency and performance for coflow-based data transfer. Many of these coflow-aware flow scheduling approaches require a centralized coordinator to perform complex per-flow bandwidth allocation. Such centralized per-flow bandwidth allocation, however, has its limitations. The main drawback is that the coordinator needs to be aware of where congestion is occurring in the network and the paths taken by each flow, which makes using these designs difficult as the location of congestion changes dynamically. In addition, coflow-based approaches require modifications to applications to use the coflow API [23].

Another direction in informing networks about the networking demands of applications is through the proactive prediction of application performance. Performance prediction through application modeling has been explored in recent works. Ernest [132] is a performance prediction framework that is designed to work with unmodified jobs and has low overhead. It does this by running a set of instances of the entire job on samples of the input, and using the data from these training runs to create a performance model. If the model is not appropriate for a particular workload, Ernest can detect this and can use small extensions to model more complex workloads. CherryPick [5] is a system that uses Bayesian Optimization to build performance models for various applications. These models are able to accurately identify the best or near-best configuration for an application with only a few test runs. CherryPick leverages information from history jobs, aiming at finding a cloud configuration (CPU, memory, number of nodes, etc.) for a single application that minimizes the cost of executing the application. Additionally, some works have delved into performance modeling as a function of

network latency. One such study employs polynomial fitting to experimental data obtained through a systematic degradation of network latency during an offline profiling phase [109, 107]. This approach helps quantify the impact of varying network latency on application performance, offering valuable insights into network optimization for different applications and workloads.

At the heart of this thesis lies a fundamental focus on best-effort-sharing approaches for bandwidth allocation within datacenter environments. The choice of best-effort methods is driven by their inherent generality and versatility. These approaches offer a robust foundation for addressing the diverse networking demands of various applications while accommodating the dynamic nature of datacenter workloads. By prioritizing best-effort techniques, this thesis aims to provide a solution that can be readily adopted and adapted to the ever-evolving landscape of datacenter networking, ensuring both flexibility and effectiveness in meeting the diverse needs of modern applications.

Chapter 3

Characterizing the Impact of Bandwidth on Applications

A diverse range of data-intensive applications coexists in today's private datacenters, including machine learning training [138, 47, 114, 81, 146, 11], SQL queries [13, 70, 136], graph processing [45, 84, 86], and big-data analytics [102, 115] co-exist. The vast majority of these applications are repetitive, distributed, and leverage parallel frameworks, such as Hadoop, Spark, Flink, and TensorFlow [122, 28, 141, 12, 1]. These frameworks follow communication models that involve sending data between servers through hundreds of connections in large amounts, causing a high demand on the network, leading to congestion and queueing delays, and affecting applications' completion time.

Datacenter operators regularly scale the capacity of the network fabric to keep up with the growth in bandwidth demands of the hosted applications; however, congestion continues to be a problem. In order to control the use of network bandwidth in the presence of congestion, datacenter networks deploy bandwidth allocation schemes and allocate bandwidth according to allocation policy. Bandwidth allocation policy defines how to distribute the link capacity among flows when congestion happens.

There has been a slew of proposals in the literature on how to improve congestion control and bandwidth allocation in datacenters and efficiently share network bandwidth among competing flows and applications. Some of these proposals focus on providing per-flow fairness [75, 30, 44] with various network-level objectives, such as minimizing per-packet latency [54] or flow completion time [7, 94, 9] using network-level properties, such as flow size or deadlines [21, 9, 94, 54, 7, 131]. Other works aim to provide isolation among tenants or applications when running in shared datacenters [50, 16, 58,

Table 3.1: Dataset size of workloads in profiling.

Workload	Dataset Size
LR (Logistic Regression)	10k samples
RF (Random Forest)	20k samples
GBT (Gradient Boosted Trees)	1k samples
SVM (Support Vector Machines)	150k samples
NI (Nutch Indexing)	100G samples
NW (NWeight)	Graph size (# of edges): 4250M
PR (PageRank)	50M pages
SQL	Two tables (# of records): 5000M and 120M
WC (WordCount)	300GB
TS (TeraSort)	280GB

17, 106, 135, 72].

A common trait of all of these proposals is that they aim to achieve some variant of max-min fairness at the flow or application level. In this chapter, we challenge this conventional wisdom and argue that max-min fairness is not the right metric to optimize for as different applications exhibit different degrees of sensitivity to the amount of network bandwidth, and hence, splitting the bandwidth equally among them may lead to sub-optimal performance.

Our analysis using a broad collection of workloads shows that the impact of network bandwidth varies across applications. Thus, different applications exhibit various degrees of *sensitivity* to bandwidth. For example, given a 56Gbps network link, reducing link capacity by 25% has no noticeable impact on a TeraSort (TS) workload, while for a Logistic Regression (LR) workload, job completion time increases by 28%.

The main contribution of this chapter is to show that enforcing network-level max-min fairness as an application-agnostic bandwidth allocation scheme can lead to poor aggregate application performance when multiple applications share the network.

3.1 Methodology

Setup: We conduct our experiments on a cluster of 8 servers. Each server runs Ubuntu 18.04 and is equipped with two 8-core Intel Xeon E5-2650v2 (Ivy Bridge) CPUs at 2.60GHz. Each CPU has 20 MB of L3 cache and two hardware threads per core, though

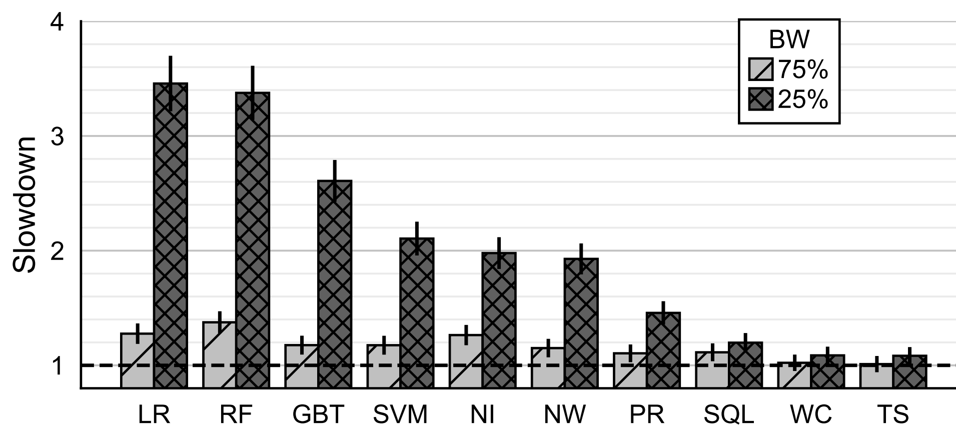


Figure 3.1: Impact of available bandwidth on the performance of workloads.

we disable SMT in our experiments. Each server has 64 GB of system memory and a single-port 56Gb InfiniBand NIC (ConnectX-3) connected on socket 0. NICs are interconnected via a Mellanox SX6036G InfiniBand switch, which supports 9 VLs (only 8 VLs are configurable).

Workloads: We use ten workloads from Intel's industry benchmarks [55] running on top of Spark and Flink. The workloads and the evaluated dataset sizes are summarized in Table 3.1.

Metric: We measure the *slowdown* compared to the execution in isolation with unthrottled bandwidth.

3.2 Sensitivity to bandwidth in applications

To assess the impact of network bandwidth on application performance, we profile a set of workloads in isolation and compute the completion time for different percentages of network bandwidth (75% and 25%).

Figure 3.1 shows the slowdown of various workloads under different percentages of available bandwidth. As the figure shows, *the degree of slowdown varies across the workloads*. For example, while LR suffers a 1.3× performance penalty when 75% of the bandwidth is available, the impact on TS's performance is negligible. With 25% of bandwidth, the slowdown of applications varies from 1.1× (TS) to 3.4× (LR), with an average of 2.1×. This analysis indicates that applications are not equally sensitive to bandwidth degradation.

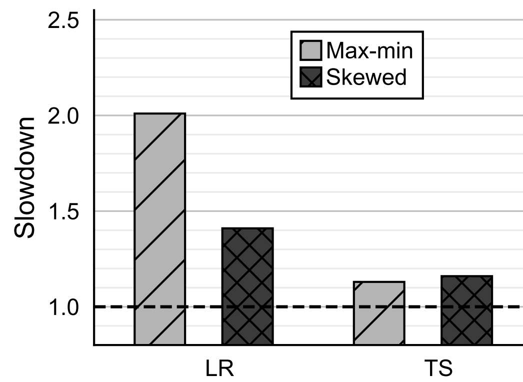


Figure 3.2: Impact of bandwidth allocation scheme on the performance of two co-running workloads.

3.3 Does flow-level fairness offer optimal performance?

The analysis in the previous section indicates that applications are not equally sensitive to bandwidth degradation. This presents an opportunity to rethink the use of max-min fairness and, instead, allocate network bandwidth based on each application's sensitivity to it. The intuition is that by providing more bandwidth to the applications that are most sensitive, their completion time can be reduced. To validate this, we run an experiment comparing the slowdown experienced by two workloads, LR and TS, when running together compared to the stand-alone execution. We run one instance of each workload on every server with a core assigned to each workload and memory equally partitioned among all workloads.
We consider two different bandwidth allocation schemes:

1. Max-min: According to the first scheme, every network component (such as NICs and switches) insures that max-min fairness is employed per flow. Therefore, each workload gets 50% of the bandwidth of shared links when both workloads use the network simultaneously.

2. Skewed: As illustrated in Figure 3.1, LR and TS exhibit different degrees of sensitivity to bandwidth. The second allocation scheme leverages this information and provides more bandwidth to LR and less to TS. In this experiment, the skewed scheme allocates 75% of bandwidth to LR and 25% to TS, i.e., the ratios shown in Figure 3.1.

Figure 3.2 illustrates the slowdown of the two co-running workloads under the different allocation regimes. In the first configuration, where per-flow max-min fairness is

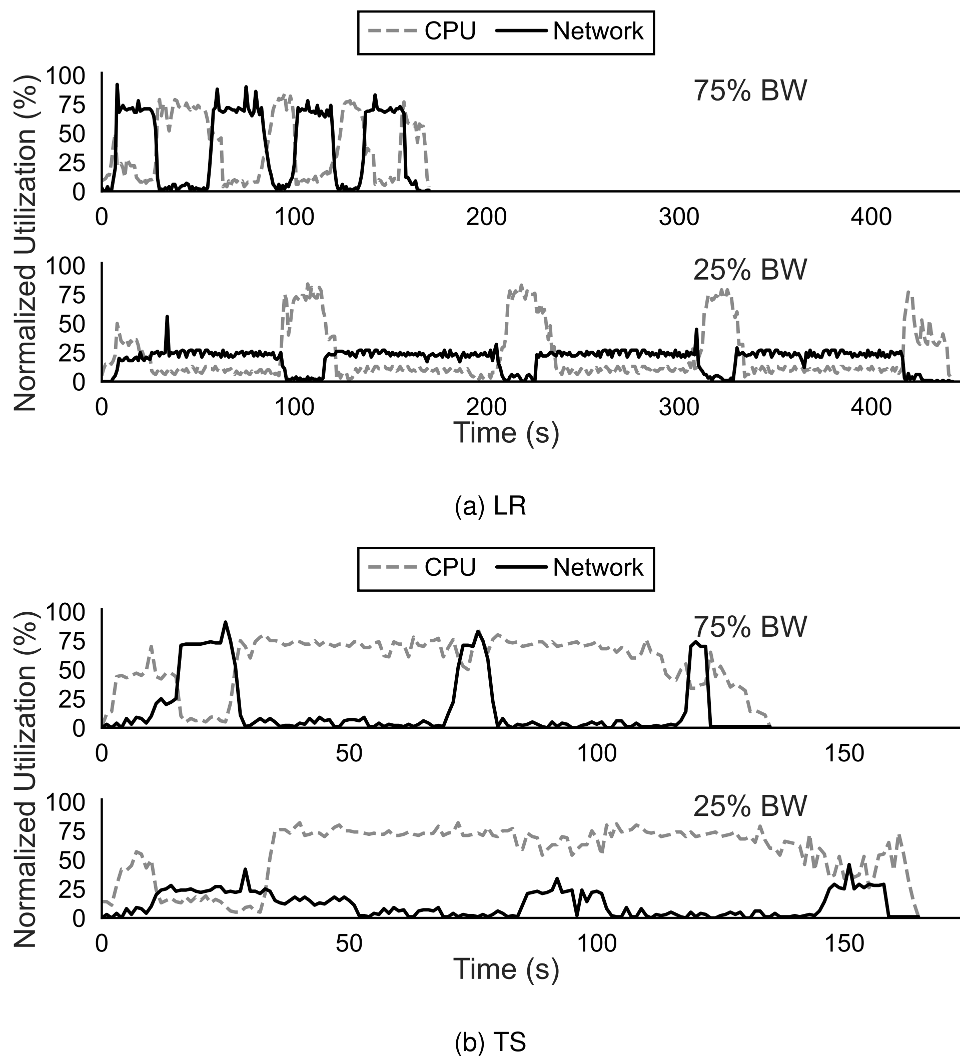


Figure 3.3: Impact of available bandwidth on resource utilization and completion time.

enforced across the cluster, LR and TS face 2.03× and 1.13× slowdown, respectively, compared to stand-alone execution. In the second configuration, in which more bandwidth is allocated to LR, the slowdown of LR decreases from 2.03× to 1.41× (62% improvement) while the slowdown of TS only slightly increases from 1.13× to 1.16× (<3% degradation). Overall, this amounts to an average slowdown decreased from 1.58× to 1.28×.

3.4 Why does the bandwidth sensitivity arise?

As a first step in addressing this question, we examine how bandwidth impacts the resource utilization of LR and TS. In this experiment, we run each application separately.

Figure 3.3 shows the timeline of normalized resource utilization for both CPUs and network with 75% and 25% of total network bandwidth available to the application. In the figure, a low network utilization with high CPU utilization implies a *computation* phase. Likewise, a high network utilization with low CPU utilization shows that the workload is in a *communication* phase.

As the figure shows, the duration of the computation phases in LR remains relatively constant when the available bandwidth is decreased from 75% to 25%. Meanwhile, the duration of the communication phases increases as bandwidth is decreased from 75% to 25%. As a result, the completion time of LR increases by 2.59× (from 172s to 447s). In contrast, TS has fewer communication phases and the workload is dominated by computation. Moreover, as Figure 3.3b shows, unlike LR, in TS, some of the data transmission is *overlapped* with computation (i.e., high network *and* CPU utilization). Thus, decreasing bandwidth has less impact on TS than LR because of higher overlapping. We observe that by decreasing the bandwidth from 75% to 25%, the completion time of TS increases by only 19% (from 138s to 165s).

Our analysis implies that it is not the duration of the communication phases, but, in fact, the fraction of communication to completion time that dictates the sensitivity of an application to network bandwidth. In other words, an application that spends a small fraction of its runtime on communication is less sensitive to bandwidth compared to another application that has a higher fraction of communication to completion time.

3.5 Implications for future application design

As we delve into the insights gained from the overlap of computation and communication phases in application performance, we find three critical lessons that can profoundly impact the design of future applications:

1. Asynchronous processing for network resilience: The first lesson centers around the importance of asynchronous processing. Applications that allow computation and communication to occur concurrently are more resilient to fluctuations in network bandwidth, as decreasing the fraction of exposed communication to completion time reduces the sensitivity to bandwidth. Future application designs should prioritize asynchronous approaches to maximize resource utilization and maintain consistent performance, even under network bandwidth availability.

2. Data localization strategies for reduced dependency on bandwidth: The second

lesson emphasizes the significance of data localization strategies. By efficiently managing data placement and movement within an application, we can significantly reduce the amount of communication, and hence, reliance on high network bandwidth. Future applications should incorporate intelligent data caching, prefetching, and local processing mechanisms to minimize the need for frequent, bandwidth-intensive data transfers.

3. Network-aware application design: A third crucial lesson is the value of network-aware application design. Applications should be equipped with the capability to adapt dynamically to changing network conditions. This adaptability includes real-time monitoring of network performance, enabling applications to make informed decisions and optimize data transfer and communication patterns accordingly. By being aware of the network's state, applications can better manage bandwidth constraints and maintain optimal performance.

These lessons underscore the importance of blending asynchronous processing, data localization strategies, and network awareness into the fabric of future application designs. Implementing these principles will enhance an application's ability to thrive in varying network environments.

3.6 Discussion

The results above are important because they demonstrate that, for these kind of workloads, *i)* equally distributing the bandwidth like in traditional max-min fairness protocols (e.g., TCP), or *ii)* focusing on reducing average *flow* completion time for individual flows (e.g., Homa [94]) or for coflows[1] (e.g., Sincronia [2]), does not necessarily result in shorter average *application* completion times. In fact, our experiments in Figure 3.2 show that by *unequally* distributing the bandwidth between LR and TS, and by *increasing* the flow completion time for the less sensitive application (TS), the average completion time of the applications is significantly reduced. Instead of improving the completion time for indirect metrics, sensitivity information of applications can be used to distribute bandwidth more effectively under congestion. Thus, in this chapter, we conclude that the sensitivity of applications should be the primary determinant of network bandwidth allocation rather than traditional network-level and application-agnostic metrics that are used at the network level.

[1] collection of related flows

Building a sensitivity-aware allocation scheme as described in Section 3.3, however, requires solving the following challenges, which we address in the following chapter:

- Sensitivity Differentiation: A sensitivity-aware solution requires a robust approach to accurately identify and capture the extent to which applications are sensitive to network bandwidth. This information is crucial in order to effectively allocate bandwidth and provide an appropriate level of service to each application.
- Dynamism: At the datacenter scale, a multitude of applications will share the network, and there will be constant changes as new applications arrive, while others terminate or migrate. In order to effectively allocate bandwidth and handle this dynamic environment, the mechanism must be able to respond quickly and efficiently to these changes.
- Practicality: In order to facilitate widespread adoption and maximize its general applicability, a bandwidth allocation scheme should not require changes to existing hardware or network protocols. This would enable the scheme to be implemented without incurring the cost and disruption of upgrading or replacing existing infrastructure.

3.7 Summary

In this chapter, we demonstrate the shortcomings in bandwidth allocation disciplines that are based on max-min fairness or shortest-flow first on a per-flow basis. We show that such allocation schemes do not effectively utilize the network in shared environments like datacenters, as they are unable to identify the bandwidth demands of applications. These approaches try to minimize the flow-level completion time, which can be in conflict with the application-level performance in shared environments; thus, they can be sub-optimal for maximizing the applications' performance. Our analysis validates that optimizing individual flows in a shared environment with bandwidth contention is inadequate to allocate bandwidth most effectively, as it is unclear how such allocation affects the performance of the associated applications. To provide the most effective allocation out of the available bandwidth, the allocation scheme should distinguish between applications that need more bandwidth for performance and those that can get away with less without drastically slowing down their completion time.

Our findings discussed in the previous paragraph show opportunities to reduce the average completion time of the applications by deploying a sensitivity-aware bandwidth

allocation scheme in datacenters. In the following chapter, we present such a bandwidth allocation scheme based on our characterization findings.

Chapter 4

Saba: Application-Aware Bandwidth Allocation Scheme

Today's private datacenters host thousands of applications, many of which are distributed and rely on a high-bandwidth communication fabric to meet their performance goals. In response, datacenter operators have deployed network protocols optimized to datacenter needs [7] and introduced fully offloaded network stacks through the use of FPGA-enabled NICs [20, 150, 34] or natively-offloaded fabrics such as InfiniBand [14]. While these improvements have brought down the inter-node communication latency to microseconds, addressing the intra-datacenter bandwidth problem has proved more challenging [64].

As explained in Chapter 3, mainstream bandwidth allocation schemes adopted in datacenters typically follow one of the two flow-level approaches: 1) either they strive to achieve *max-min fairness* by equally partitioning network bandwidth across flows, or 2) they opt for an approximation of the *Shortest Remaining Processing Time (SRPT)* algorithm, aiming to minimize the completion time of applications by prioritizing those with the shortest remaining flows. Instead, we propose a new metric called *bandwidth sensitivity*, which captures the impact of the network bandwidth on the completion time for a specific application. Our analysis in the previous chapter shows that different bandwidth-intensive applications exhibit various degrees of bandwidth sensitivity. For example, given a 56Gbps network link, reducing the link capacity to 25% leads to a 3.4× increase in the completion time of a Logistic Regression (LR) workload, while the completion time of a PageRank (PR) workload increases by a factor of 1.4×.

This observation is at the core of Saba, a novel application-aware allocation scheme that distributes network bandwidth to applications proportionally to their bandwidth

sensitivity. Saba computes the bandwidth sensitivity of applications in advance and leverages this information at runtime to derive the weights used in switches' priority queues to enforce the desired bandwidth allocations in a work-conserving fashion.

We put particular effort into the design of Saba to make it not only effective but also practical for use in real-world datacenter environments. With this in mind, we make a conscious effort to minimize the resources required during the profiling process and the number of queues used in switches. By optimizing these elements, we aim to make Saba as seamless and efficient as possible for existing datacenter deployments. Saba does not mandate any changes to deployed congestion-control protocols and it is fully compatible with existing switches and NICs while requiring only a lightweight shim layer ($\sim$350 LOCs) installed at the end hosts.

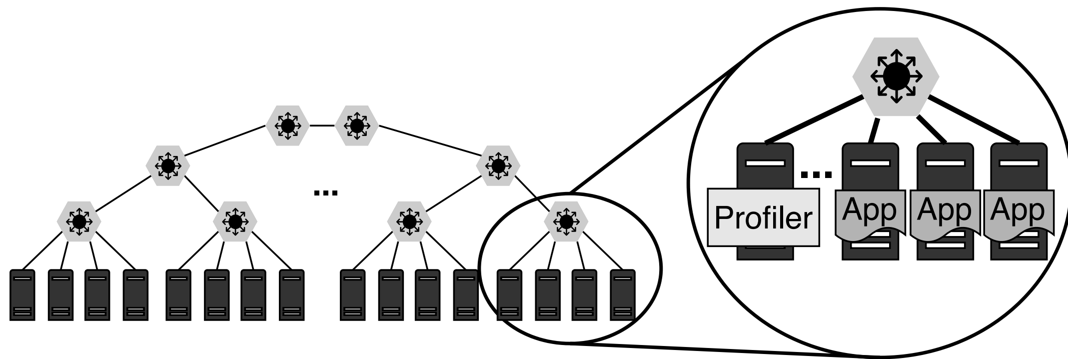
We evaluate Saba in a 32-server InfiniBand cluster across a broad set of workloads. We also simulate a Saba deployment with 1,944 servers to assess its performance at scale. Our experimental and simulation results show that Saba effectively improves the average performance of the co-running applications sharing the network, compared to both existing and *ideal* implementations of per-flow max-min fairness, as well as state-of-the-art SRPT-based approaches that work at the flow or coflow level.

The main contributions of this chapter are the following:

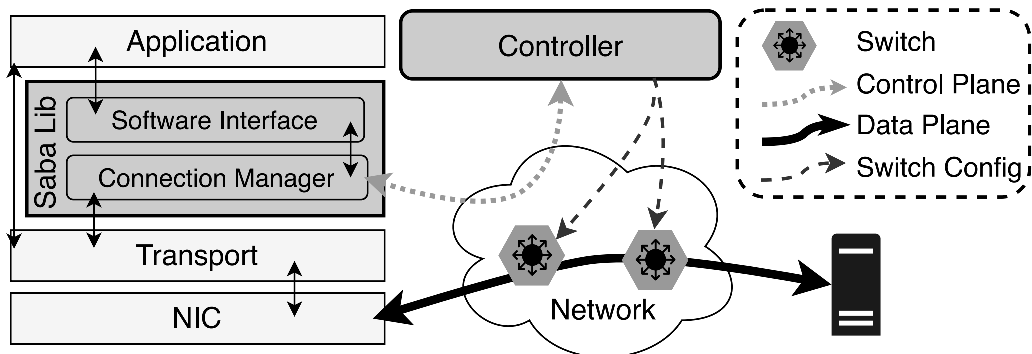
- We introduce the notion of bandwidth sensitivity as a guiding principle to allocate bandwidth among applications and show how this can be learned through profiling.

- We present Saba, our application-aware bandwidth allocation framework, which relies on bandwidth sensitivity for bandwidth allocation. We compare Saba against a baseline using InfiniBand congestion control and demonstrate that Saba can reduce the completion time for bandwidth-sensitive jobs by up to 3.94$\times$ while only marginally impacting a few of the bandwidth-insensitive jobs (1-5% slowdown), improving the average completion time by 1.88$\times$.

- We show in simulations that similar benefits also hold at scale and against an ideal implementation of max-min fairness, obtaining up to 1.79$\times$ speedup (3% slowdown in the worst case) with an average improvement in completion time by 1.27$\times$.



(a) The offline profiler deploys applications and profiles them.



(b) The controller and Saba library.

Figure 4.1: An overview of the main components of Saba.

4.1 Saba overview

We introduce Saba, an application-aware bandwidth allocation scheme that aims to maximize performance across non-interactive applications that share the network in a datacenter. At the heart of Saba is a metric called *bandwidth sensitivity*, which reflects the effect of network bandwidth on application performance. We define bandwidth sensitivity for a given application as *the degree of performance degradation caused by a reduction in the availability of bandwidth.* Saba's key idea is that providing more bandwidth to applications that are more sensitive to bandwidth can reduce their completion time, without compromising the performance of bandwidth-insensitive applications.

Saba consists of three main components: an offline profiler, a controller, and a library (Figure 4.1). The *profiler* determines the bandwidth sensitivity of applications by profiling them in advance (Figure 4.1a). The *controller* uses the bandwidth sensitivity

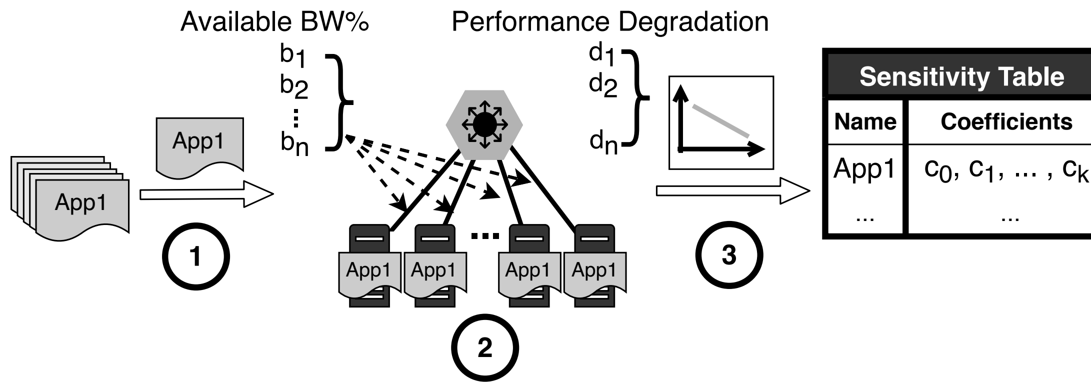


Figure 4.2: Details of the offline profiler.

information provided by the profiler to calculate the bandwidth share of applications in a way that minimizes the average slowdown across applications and orchestrates the network switches to enforce bandwidth. Saba expects compliant applications to be registered at launch. The Saba *library* provides a software interface for applications to communicate with the controller and conduct the registration (Figure 4.1b).

While Saba dynamically partitions network bandwidth for Saba-compliant applications, non-Saba-compliant applications (e.g., control or latency-critical services) can co-exist on the same network. To support this co-existence, datacenter operators can statically allocate a queue for non-Saba-compliant applications on switches and reserve a portion of the network bandwidth for them. Such a queue-based separation effectively isolates flows of non-Saba-compliant applications, preventing interference between them and Saba-compliant applications. Thus, it allows Saba-compliant applications to benefit from the dynamic bandwidth allocation without negatively impacting the performance of other applications running on the same network. While we have implemented Saba on top of an InfiniBand architecture, our design does not make any assumptions about the underlying network layers; e.g., our implementation can be easily ported to Ethernet networks. The following sections provide details of the aforementioned components of Saba (*profiler*, *controller* and *library*), as well as details of our InfiniBand implementation.

4.2 Profiler

Saba borrows the idea of using a-priori profiling of applications to make resource allocation decisions [137, 5, 132, 109]. Saba's offline profiler performs ahead-of-time profiling on applications to measure their bandwidth sensitivity based on the perfor-

mance degradation caused by limited network bandwidth. The profiler uses *application completion time* as the metric of performance. Completion time is commonly used to evaluate the performance of data-intensive applications [73] and can easily be determined at the system level by recording the start time and end time of an application.

4.2.1 Profiling process

Figure 4.2 depicts the profiling process managed by the profiler. To profile a given application *App1*, the profiler first deploys the application multiple nodes ①. In order to measure the performance degradation of the application caused by changes in available bandwidth, the profiler runs the application n times. In each run, the profiler limits the bandwidth of all NICs to a certain percentage of link capacity from $BW = \{b_1, b_2, \ldots, b_n\}$ and measures the completion time of the application ②. Next, for each measured completion time, the profiler determines the performance degradation by calculating the *slowdown*, i.e., the ratio of completion time under a given percentage of bandwidth to the completion time with unthrottled bandwidth. The output is $D = \{d_1, d_2, \ldots, d_n\}$, a set of performance degradation values, each of which represents the impact of the corresponding percentage of bandwidth on the performance of the application. The profiler uses $Samples = \{(b_1, d_1), (b_2, d_2), \ldots, (b_n, d_n)\}$ to generate a *polynomial regression model* and establish the relationship between the bandwidth and the slowdown of the application ③. We refer to this regression model as the *sensitivity model*. The sensitivity model of *App1* can be represented as follows:

$$D_{App1}(b) = c_0 + c_1 b + c_2 b^2 + \cdots + c_k b^k = \sum_{i=0}^{k} c_i b^i \tag{4.1}$$

where k is the *degree of polynomial*. The profiler determines the value of the coefficients, $\hat{C} = \{c_0, \cdots, c_k\}$, by fitting the polynomial to the samples, and records the coefficients in the *sensitivity table*. Saba uses this table in its controller for bandwidth allocation (Section 4.3).

4.2.2 Accuracy of sensitivity models

In order to evaluate the goodness-of-fit and accuracy of a model, we use R^2 (coefficient of determination) [79]. R^2 quantifies the fraction of the slowdown trend that the model is able to explain. $R^2 = 1$ implies that the model explains all the observed slowdowns in response to bandwidth. A model with $R^2 = 0$ does not explain any of the observed

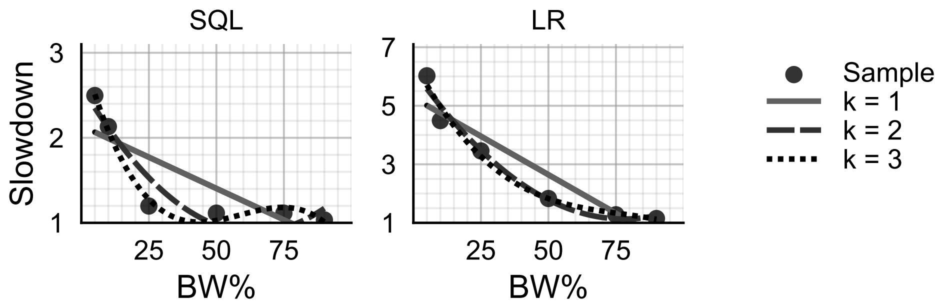


Figure 4.3: Sensitivity models of SQL and LR workloads with various degrees of polynomial (k).

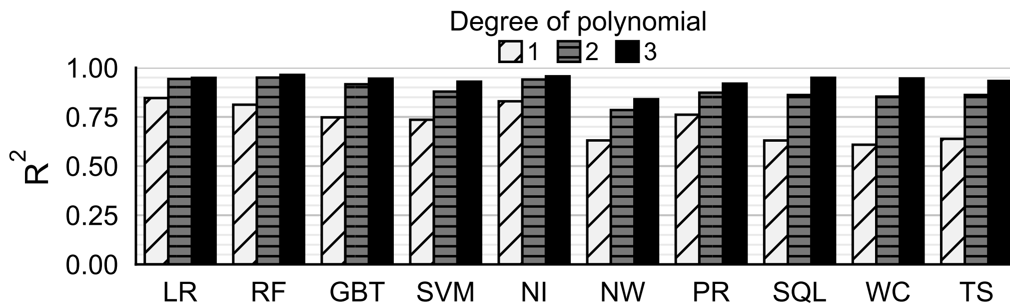


Figure 4.4: Impact of degree of the polynomial on the accuracy of sensitivity models.

slowdown trend. The accuracy of the sensitivity model generated by the profiler depends on the degree of the polynomial, and on the differences between settings at runtime as compared to profile time. These settings include the size of the dataset and the number of nodes. We next discuss the effect of each of these parameters on R^2 while we show their impact on completion time in Section 4.6.2.

Degree of polynomial: The degree of polynomial determines the ability of the model to capture any non-linearity in the relationship between bandwidth and performance degradation of an application. Non-linearity within the sensitivity curve relating bandwidth to the degradation of performance in a given application indicates that changes in bandwidth do not consistently and proportionally influence performance outcomes. This non-linear behavior is rooted in the intricate interplay between the computation and communication phases of the application. As explained in Section 3.4, the architecture of the application may allow for concurrent computation and communication, effectively concealing some of the communication overheads. Consequently, restricting the bandwidth might not always lead to a proportional increase in completion time, up to a certain threshold. This is due to the ability of the application to efficiently

overlap these phases, mitigating the direct impact of bandwidth reduction on overall execution time within a certain range. Thus, it is essential to accurately capture this non-linearity to predict how changes in bandwidth impact application performance. We assess the degree of non-linearity for the studied workloads by profiling them as discussed in Section 4.2.1 (see the complete methodology in Section 4.6.1). Figure 4.3 plots the profiling samples for two studied workloads, SQL and LR, along with the sensitivity models for three different degrees of polynomial.

As Figure 4.3 shows, the non-linearity of the samples varies across the two workloads. For SQL, the non-linearity is high – as bandwidth is decreased from 100% to 25%, performance degrades by a mere 1.2×; however, as the bandwidth is further reduced to 10%, performance drops by 2.2×. In contrast, LR experiences a higher performance degradation throughout the bandwidth range but exhibits a more linear correlation between performance and bandwidth. Performance of LR degrades by 1.3×, 3.4×, and 4.5× as bandwidth is decreased to 75%, 25%, and 10%, respectively. The figure clearly shows that a model with a first-degree polynomial is unable to accommodate the data for SQL, and while a second-degree polynomial perfectly fits the data for LR, a third-degree polynomial is needed for a good fit of SQL's data.

We next study the accuracy of the sensitivity models based on different degrees of the polynomial. Figure 4.4 shows that as the degree of polynomials increases, R^2 of the sensitivity models also increases. The sensitivity models of all workloads show accuracy above 0.60 using the first-degree polynomial ($k = 1$). We observe that some workloads enjoy a big jump in their accuracy by using higher degrees of polynomial in their sensitivity model; e.g., R^2 for the sensitivity model of SQL increases from 0.63 to 0.96 as we increase the degree of polynomial of the model from 1 to 3. For other workloads, lower degrees of polynomial provide models with high accuracy; e.g., the sensitivity model of LR generated with $k = 1$ attains an accuracy of 0.84. Using $k = 2$ increases the accuracy of the model to 0.94, while $k = 3$ provides only a small additional improvement to LR and R^2 increases to 0.95. We conclude that the degree of polynomial directly affects the accuracy of sensitivity models.

Application dataset size: In practice, it may be challenging to estimate dataset size accurately in the profiling phase. It may also be desirable to run profiling with a smaller dataset size than in deployment to reduce profiling time and cost. Thus, the size of the dataset at runtime may differ from that used by the profiler. To analyze how the accuracy of each model depends on the dataset size, we conduct a study to estimate the accuracy of the sensitivity model of each workload when the size of datasets used at

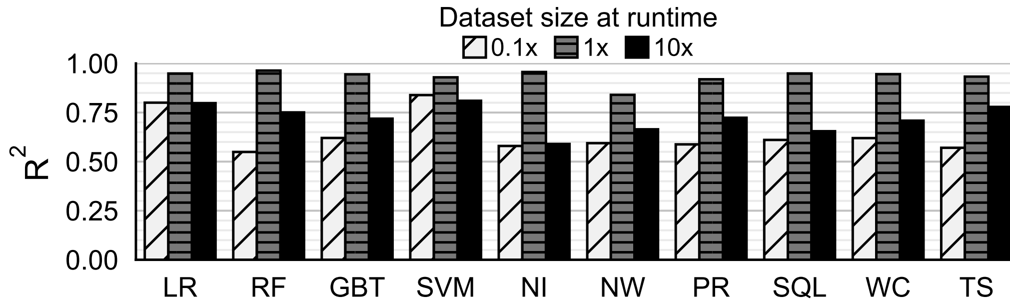


Figure 4.5: Impact of dataset size at runtime on the accuracy of sensitivity models.

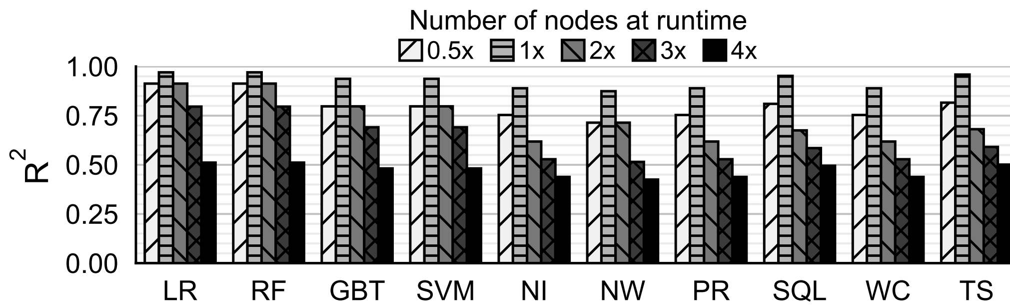


Figure 4.6: Impact of the number of nodes at runtime on the accuracy of sensitivity models.

runtime differs from the profiling datasets. In this study, all models are generated with the polynomial of degree three ($k = 3$).

Figure 4.5 shows the impact of the runtime dataset size on the accuracy of sensitivity models of the studied workloads in extreme cases in which the dataset size at runtime is ten times smaller (0.1x) or larger (10x) than the dataset used by the profiler (1x). We observe that while using datasets with sizes of 0.1x and 10x reduces the accuracy of the models, all models obtain R^2 above 0.55. SVM experiences the most negligible impact as the accuracy of its model reduces from 0.92 to 0.83 (dataset size 0.1x) and 0.81 (dataset size 10x). Nutch Indexing (NI) experiences the highest impact as the accuracy of its sensitivity model decreases from 0.95 to 0.57 (dataset size 0.1x) and 0.59 (dataset size 10x). We conclude that for the studied workloads, an order of magnitude difference in dataset size between profiling and runtime attains R^2 above 0.55, indicating that the sensitivity model retains good predictive power despite the change in dataset size.

Number of nodes: Similar to the dataset size, the number of nodes running a distributed application may not be the same as the number of nodes used by the profiler. For instance, limiting the number of nodes used in the profiling phase may be desirable

to contain the costs of profiling. To study the impact of the number of nodes at runtime on the accuracy of sensitivity models, we compare the R^2 of models of the workloads across various numbers of nodes at runtime, ranging from 0.5x to 4x of the number of nodes used by the profiler (8 nodes). All models use $k = 3$.

As Figure 4.6 shows, the sensitivity models of all workloads maintain an accuracy above 0.50 when the number of nodes at runtime ranges from 0.5x to 3x. The sensitivity model of NWeight (NW), delivers the lowest observed accuracy $R^2 = 0.51$ when the number of nodes is 3x. When increasing the number of nodes at runtime to 4x compared to profiling, we observe that R^2 drops for most models to below 0.50; the exceptions are LR, Random Forest (RF), and TS. We conclude that the number of nodes is a crucial factor governing the accuracy of the sensitivity models.

While evaluating the accuracy of models in terms of R^2 is helpful in understanding the predictive power of models, our end goal is to assess the impact of errors on the performance of Saba. We quantify the performance of Saba with varying degrees of polynomial, dataset size, and the number of nodes in Section 4.6.2.

4.3 Controller

Saba relies on a controller to perform bandwidth allocation and orchestrate switches for bandwidth enforcement. To conduct the allocation and enforcement, the controller requires the following information: 1) which applications are Saba-compliant, and 2) the source and destination of each connection for each Saba-compliant application. The controller needs information about the source and destination of a given connection to determine the switches along their path[1]. Applications explicitly or transparently send the above information to the controller via a software interface provided by the Saba library (details are in Section 4.4). The controller collects the above information from applications and determines the bandwidth share of each application at runtime. To calculate the bandwidth share of applications, Saba uses the bandwidth sensitivity information in the sensitivity table provided by the profiler. The controller assigns the allocated bandwidth to applications and configures the switches to enforce the bandwidth shares.

[1]If the underlying network layer supports multipathing, the controller determines switches along all paths between the source and destination.

4.3.1 Bandwidth calculation and assignment

Saba allocates bandwidth for applications that have registered themselves via the Saba library and are actively using the network. By tracking the active applications through the Saba library, the controller has global information about the paths of Saba-compliant flows passing through switches in the network (see Section 4.3.4 for details on scalability). The controller uses this information combined with the profiling result in the sensitivity table and determines the percentage of bandwidth to be allocated to the flows from each application at each switch output port in a way that *minimizes the total slowdown across applications*.

For a given set of applications $\hat{A} = \{a_1, a_2, \ldots, a_n\}$ sending flows to a given switch output port, weight w_i represents the percentage of bandwidth allocated to application a_i at that port. The goal is to find a set of weights to minimize the total slowdown across applications. To do so, the controller uses the sensitivity models generated by the profiler to predict the slowdown of each application. $\hat{D} = \{D_1, D_2, \ldots, D_n\}$ is the set of sensitivity models corresponding to $\hat{A}$, each of which generated via Eq 4.1. The controller calculates the weights $\hat{W} = \{w_1, \ldots, w_n\}$ for applications at the given switch output port as follows:

$$\begin{aligned} \hat{W} &= \arg\min_{\vec{W}} \sum_{i=1}^{n} D_i(w_i) \\ \text{subject to} & \sum_{i=1}^{n} w_i = C_{Saba} \end{aligned} \tag{4.2}$$

where C_{Saba} is the percentage of link capacity that the operator has reserved for Saba-compliant applications. The controller uses these weights to configure the given output port of the switch and enforce bandwidth.

4.3.2 Bandwidth enforcement

Saba enforces the allocated bandwidth at network switches, utilizing the available rate-limiting mechanisms at the transport layer. This approach decouples the bandwidth allocation and enforcement from congestion management, leading to a cleaner design. Thus, Saba does not need to calculate and limit transmission rates at the endpoints and leaves the rate-limiting to the congestion-control protocol.

Bandwidth enforcement at switches works in Saba without any modification to existing switches, as long as the network layer supports the following requirements:

1) service differentiation through Priority Levels (PLs), and 2) per-port queues in switches. Service differentiation is required to differentiate flows coming from different applications. Per-port queues in switches are needed to enforce bandwidth by assigning flows with weights to queues. Moreover, Saba assumes that switches implement the Weighted Fair Queuing (WFQ) scheduling algorithm to schedule the packets inside the per-port queues in proportion to the weights of queues. WFQ is work-conserving, meaning that other applications may utilize the remaining bandwidth quota if one application does not use some or all of its share. Furthermore, WFQ is not subject to starvation, meaning that all flows progress and are eventually transmitted, which is an advantage for a bandwidth allocation scheme. Fortunately, modern switches used in datacenters support both service differentiation and per-port queues and implement variations of WFQ [14, 94].

In Saba, the controller assigns a PL to flows coming from each application with the help of the Saba library (Section 4.4), and maps each PL to a queue in the switch port. The controller configures each switch output port with a set of calculated weights $\hat{W} = \{w_1, \dots, w_n\}$ determined via Eq (4.2) and assigns each weight to the corresponding queue, and each switch services flows inside queues using the WFQ scheduling algorithm. Note that this approach assumes that switches have an unlimited number of queues, thus using an idealized one-to-one mapping of applications to queues. However, existing switches have a limited number of queues [14, 94]. We next discuss how Saba addresses this issue.

4.3.3 Mapping applications to queues

While an ideal implementation of Saba should support a one-to-one mapping between applications and queues, modern network architectures support only a limited number of PLs and queues. The number of PLs is determined by the Quality of Service specification of network technology, whereas the number of queues in NICs and switches varies across different hardware implementations. For instance, InfiniBand and Ethernet support 16 and 8 PLs, respectively [14, 94], and a typical datacenter-grade (InfiniBand or Ethernet) switch supports 4-8 queues [14, 94, 9]. To overcome this limitation, Saba performs two layers of mapping to translate applications to queues by mapping applications to PLs and mapping PLs to queues.

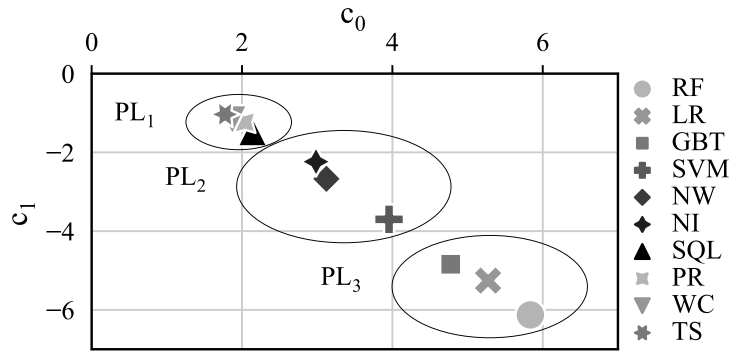


Figure 4.7: Application-to-PL mapping for the studied Spark workloads ($S = 3$). Groups of workloads are circled.

4.3.3.1 Application-to-PL mapping

At the datacenter scale with hundreds to thousands of running applications, a one-to-one mapping between applications and PLs is infeasible due to the limited number of PLs. To address this limitation, Saba groups applications according to their bandwidth sensitivity using the K-means clustering algorithm [85]. The controller takes a set of registered applications and the coefficients of their sensitivity models as input, creating S groups from them, where S is the number of PLs. The centroid of each group represents the sensitivity of that group. Saba assigns each group a PL and uses the sensitivity of the groups in the PL-to-queue mapping.

Figure 4.7 shows an example of application-to-PL mapping for the studied workloads with 3 PLs. In this example, the sensitivity of workloads is modeled with the first-degree polynomial (i.e., $k = 1$) and represented by two coefficients, c_0 and c_1 (as described in Eq 4.1). The controller groups the applications using the K-means algorithm and assigns PL_1 to TS, WordCount (WC), PR, and SQL, as the coefficients of their models are close together; meanwhile, the controller assigns SVM, NI, and NW to PL_2 and PL_3 to LR, RF, and GBT.

4.3.3.2 PL-to-queue mapping

The controller needs to map PLs to queues to complete the application-to-queue translation. This task, however, is complicated by the fact that flows in a given PL may share different links along their paths, thus resulting in a different set of flows mapping to the PL in different switches. Consequently, the weight of a PL may vary across the switches. Additionally, the number of queues in different switches varies as the capa-

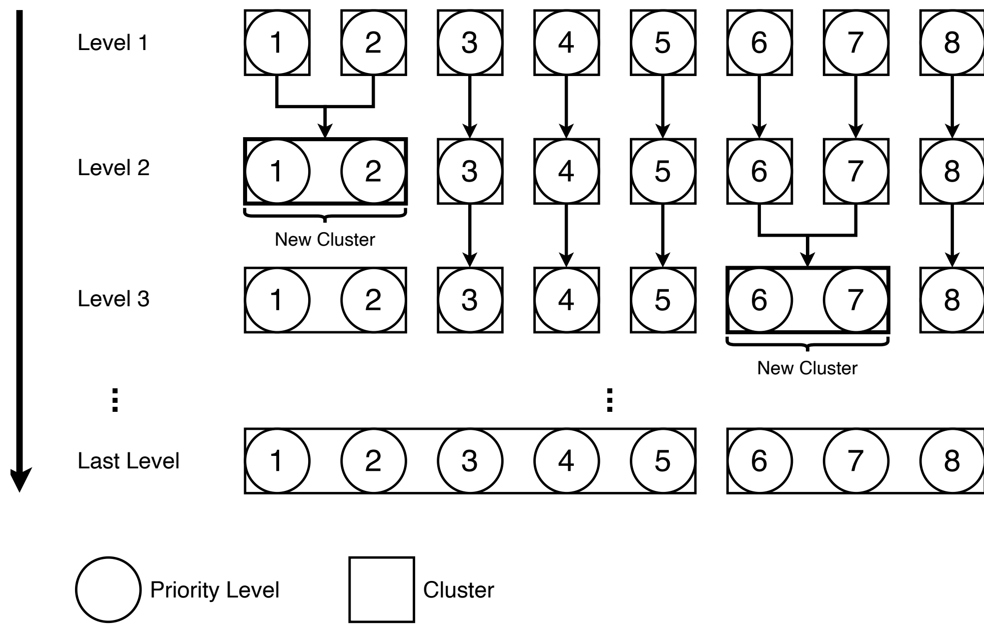


Figure 4.8: An example of clustering 8 PLs, assuming that the minimum number of queues supported in the switches is 2. Circles and boxes represent PLs and clusters, respectively. The controller clusters PLs based on the closeness of bandwidth sensitivity of associated applications in a hierarchical manner.

bility of switches is not necessarily the same. Thus, PL clustering must be performed individually at each switch output port to map PLs to queues.

To address this problem, Saba must maintain multiple PL clusters and PL-to-queue mappings and choose the appropriate mapping for each switch port at runtime. To this end, Saba introduces a hierarchical approach: (1) to cluster PLs, Saba uses a hierarchical clustering scheme to preserve the information of all possible combinations of PL clustering hierarchically; (2) at runtime, Saba finds the best clustering from the hierarchy for each switch output port and uses it for bandwidth allocation. We next detail each of these tasks.

PL clustering: To efficiently cluster PLs hierarchically, we employ the fast hierarchical clustering algorithm [95]. This algorithm, commonly utilized in data analysis and machine learning, excels at swiftly grouping similar data points into clusters, emphasizing its time-effective performance. Saba performs the PL clustering offline as follows:

(a) *Initialization:* In the first level, the controller starts by treating each PL as its own cluster, thus assigning each PL to a separate cluster.

(b) *Calculate pairwise distances:* In the next step, the controller computes the pairwise

distances (i.e., similarities) between all clusters. To do so, the controller uses the coefficients of the models as their coordinates, and Euclidean distance as the similarity metric in the distance computation.

(c) *Merge closest clusters:* In subsequent iterations, the controller merges the two closest clusters from the previous level to create a new cluster. The coefficients of the model for the new cluster are determined by the coordinates of the Euclidean midpoint (i.e., average linkage) of the corresponding coefficients of the two merged clusters. Each iteration reduces the total number of clusters by one. This merging process continues until the number of remaining clusters equals the minimum number of queues in switches (the last level).

Figure 4.8 depicts the PL clustering performed by the controller. The figure assumes 8 PLs are available, and the number of queues is 2.

Finding the best clustering: At runtime, the controller finds the appropriate PL clustering as follows: Assuming that a switch supports Q per-port queues, given a set of PLs whose flows pass through a switch output port, the controller determines the optimal cluster at that port by following these steps:

(a) Start from level 1 of the hierarchy.

(b) In the current level, if all PLs are grouped into at most Q clusters, go to (c); otherwise, go to the next level and repeat (b).

(c) Map each cluster of PLs to a queue.

Once PLs are mapped to queues, the controller assigns the sum of the bandwidth allocated to applications (Eq 4.2) associated with each queue as the weight of that queue.

Figure 4.9 illustrates an example of PL-to-queue mapping for two switches. In this example, we assume that each switch supports two queues per port. Flows from application *App1* pass through link 1 and link 2, and share each of the two links with flows from different sets of applications; consequently, flow from an application *App1* receives different weights at two switches. The controller maps PLs to queues at each switch port using the information from the PL clustering in Figure 4.8. At port *P1*, there are two PLs; and according to level 1 in Figure 4.8, PLs are in two separate clusters. The controller maps PL 1 to queue 1 and PL 4 to queue 2. At port *P1*, there are three PLs. The controller finds that in level 2, PL1 and PL2 form one cluster, and PL3 is in another cluster. The controller maps PL1 and PL2 to queue 1, and PL3 to queue 2.

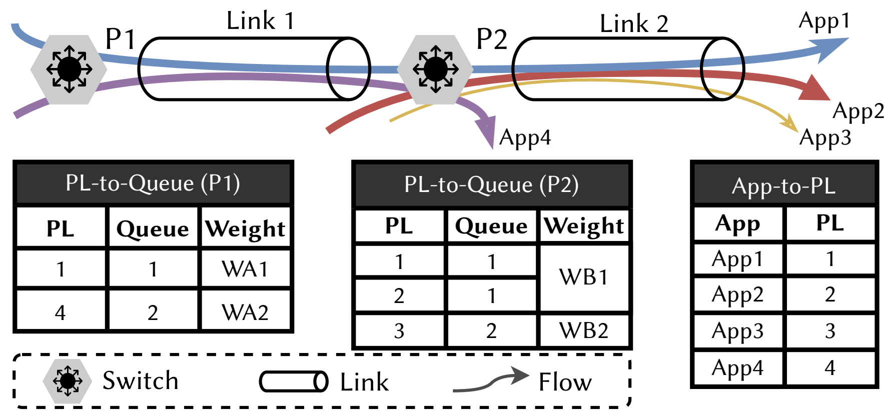


Figure 4.9: Example of PL-to-queue mapping for two switches at runtime.

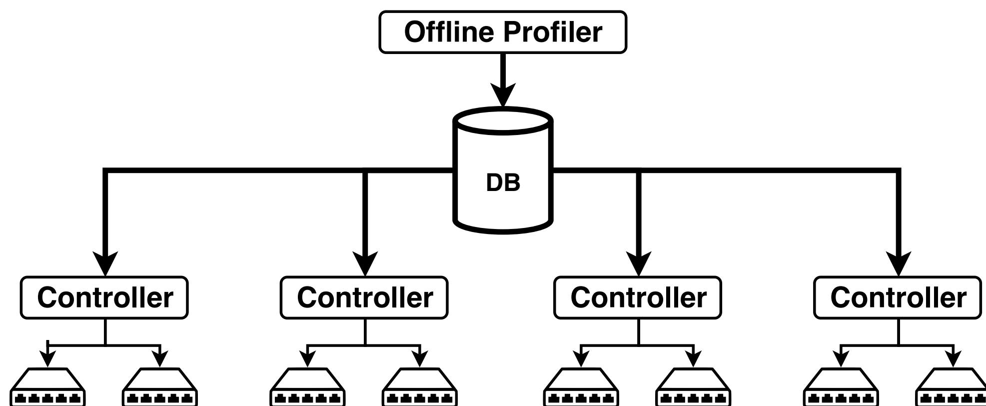


Figure 4.10: High-level overview of a distributed deployment of Saba.

4.3.4 Centralized vs distributed controller

So far, the discussion has implicitly assumed a centralized controller that maintains the global state of application-to-PL and PL clustering operations, as well as the state of each switch, including flows passing through the switch and the current switch configuration. Such a centralized controller updates the application-to-PL mapping and performs PL clustering when a new application is registered or a running application is deregistered. Furthermore, every time a connection is created or destroyed, the centralized controller performs a new PL-to-queue mapping and updates the configuration of switches on the path of that connection. Naturally, a centralized controller represents a single point of failure. In addition, calculating the bandwidth for every application at the scale of a large cluster or a datacenter may result in the centralized controller bottlenecking performance.

An alternative to the centralized controller is a distributed one (Figure 4.10). Eq 4.2 indicates that the bandwidth calculation for applications on a given output port is

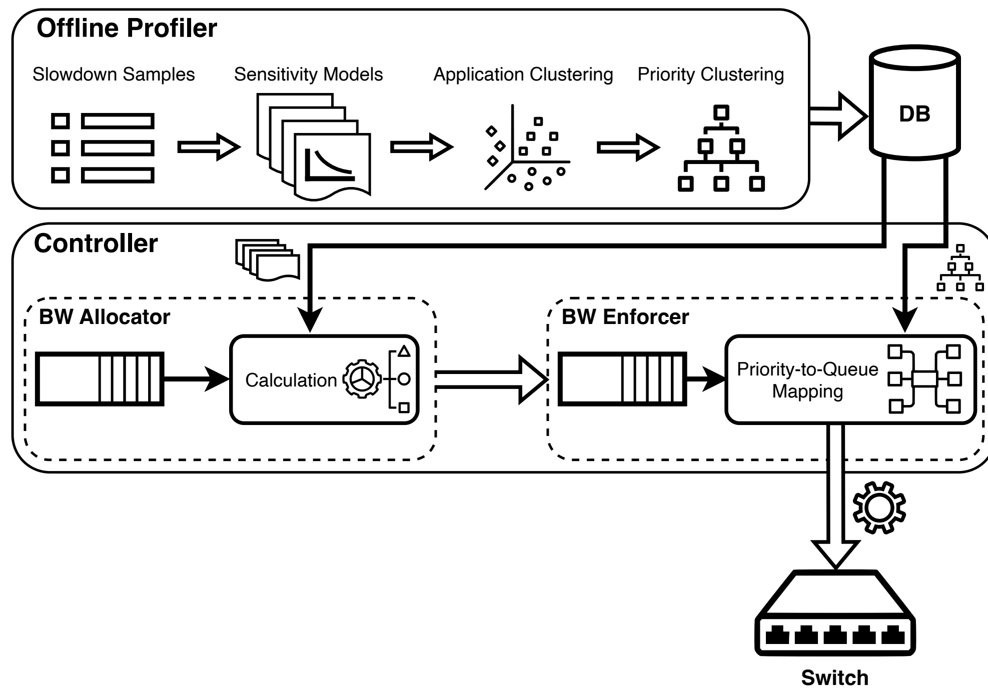


Figure 4.11: High-level overview of the workflow between the profiler and a distributed controller.

independent of other switches, presenting an opportunity to distribute the controller's logic. In such a distributed design, each controller is responsible for a group of switches and maintains the record of applications sending flows to the associated switches. In order to enforce the bandwidth, the controllers fetch the application-to-PL mapping and the PL clusters from a database. The Saba library informs the closest controller when an application requests a new connection. This controller performs bandwidth allocation and enforcement while communicating with the following controller on the path of the connection to inform it about the added connection. Each controller does the same until all switches on the path of the new connection are configured. Figure 4.11 illustrates the workflow of Saba in a such distributed deployment.

As noted above, the distributed controller design requires a database containing the outcomes of application-to-PL mapping and PL clustering operations. *The profiler* updates the database after performing the application-to-PL and PL clustering operations whenever a new application is profiled. Existing replication techniques can be used to replicate the database to increase reliability, availability, and scalability. To maximize performance, database instances can be co-located with the controller nodes.

Table 4.1: Saba software interface description.

Function	Description
`saba_app_register`	Register and receive a tag
`saba_conn_create`	Create a connection with the tag
`saba_conn_destroy`	Destroy the connection
`saba_app_deregister`	De-register the app

4.4 Saba library

Applications that wish to be Saba-compliant must register themselves at runtime via the Saba library and provide the controller with information about their networking connections. This library consists of two components: the connection manager and the software interface.

4.4.1 Connection manager

The connection manager performs two tasks: communicating with the controller and creating connections. For a given application, the connection manager communicates with the controller to register the application and receives a PL from the controller. When the application requests the creation of a connection, the connection manager creates a new connection and assigns the PL to the connection. During data exchange (send or receive), packets from all connections associated with the application carry the assigned PL. When the application creates or destroys a connection, the connection manager informs the controller about the creation or destruction of connections, leading to new bandwidth allocations in the controller.

4.4.2 Software interface

Saba features a simple software interface to communicate with the controller (Table 4.1). Figure 4.12 illustrates the interactions between the interface, the controller, and a given switch when each function from the interface is called. An application request registration at start time ①. Saba library informs the controller via the connection manager ②. In response to the registration request, the controller returns a PL (generated by the application-to-PL clustering) to the connection manager to be used for future connections ③. In order to create a connection, the application sends a request ④. In the

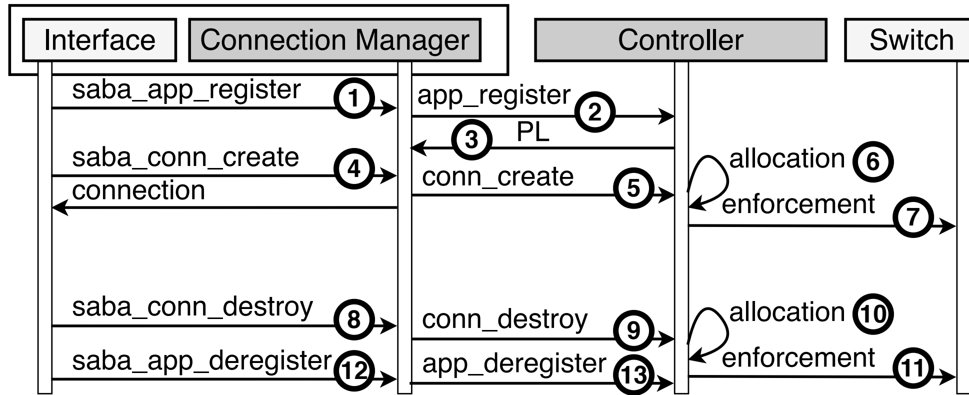


Figure 4.12: High-level overview of interactions between the software interface, the connection manager, the controller, and a network switch.

Saba library, the connection manager uses the acquired PL to create the connection and returns the connection to the application. Meanwhile, the connection manager informs the controller about the new connection (5). The controller considers the new connection and its associated PL for a new allocation (6) and enforces the allocation by updating the weights of the queues in the switches (7). Once the application no longer needs the connection, it destroys the connection (8). The connection manager informs the controller about the finished connection (9). This leads to a new allocation (10), and the controller updates the weights of queues in the switches (11). Before quitting, the application requests for deregistration (12), and the connection manager informs the controller to deregister the application (13).

4.5 Implementation

This section provides the details of the implementation of the three components of Saba, *offline profiler*, *controller* and *Saba library* for InfiniBand networks.

4.5.1 Profiler

The profiler executes the application multiple times on a set of dedicated nodes; in each run, it adjusts the amount of network bandwidth available to the application and measures the application's completion time. Our current implementation considers the following percentages of link bandwidth: $5\%, 10\%, 25\%, 50\%, 75\%$ 90% and 100%, which are enforced by a token bucket [77] rate limiter in the InfiniBand driver [98].

4.5.2 Controller

Path Detection: As explained in Section 4.3, the controller receives information about the sources and destinations of each connection from the Saba library. Saba leverages the *infiniband-diags* package provided by OpenFabric [98] and gets the forwarding tables of switches in the network to detect the path of each connection.

Weight Calculation: Saba uses Sequential Least Squares Programming (SLSQP) algorithm from NLopt library[60] to solve the optimization problem in Eq 4.2 and calculate the weights for the flows passing through a given port.

Bandwidth Enforcement: As explained in Section 4.3.2, the controller requires the following from the network to enforce bandwidth: *i)* service differentiation and *ii)* per-port queues in switches with the WFQ scheduling policy. InfiniBand supports service differentiation in its transport protocol and features per-port queues. Further, InfiniBand switches implement a Weighted Round Robin scheduling discipline to approximate WFQ. InfiniBand offers service differentiation by introducing two concepts: *Service Levels* and *Virtual Lanes* [14].

1) Service Levels (SLs): InfiniBand supports 16 priority levels, called Service Levels (SLs). SLs are exposed to the developer and can be used to create connections. Once a connection has been created using an SL, the header of packets from the connection will carry the SL through the network.

2) Virtual Lanes (VLs): InfiniBand divides a physical link into different logical communication links, called Virtual Lanes (VLs), and allocates a queue for each VL at the output ports of switches and NICs. For each VL, InfiniBand provides buffering, flow-control, and congestion management. Switches and NICs handle packets inside VLs in each scheduling turn according to a table that maps SLs with their associated weights to VLs. This table is configurable at every switch and NIC by the datacenter operator. With the current InfiniBand specification, each switch or NIC must support between 2 and 16 VLs [14].

Saba uses SLs to differentiate applications and enforces bandwidth by dynamically setting the VLs' weights at switches.

4.5.3 Saba library

The connection manager, implemented with just 350 LOC, uses RPC operations for all control-plane activities. The connection manager creates the low-level InfiniBand connections using *ibverbs* library and assigns SLs to them. In order to register workloads

and communicate with the controller, we modified the existing job submission and shuffle manager in Spark and Flink to invoke Saba's interface functions. However, the individual workloads required *no* modification to support Saba.

4.6 Evaluation

We evaluate Saba using experiments on (1) a 32-server InfiniBand testbed with a suite of workloads and (2) a simulated 1,944-server cluster with a set of synthetic workloads.

4.6.1 Methodology

The main goal of the experiments is to evaluate the impact of Saba on the performance of workloads as compared to a baseline implementing max-min fairness.

Baseline: We use InfiniBand as our baseline, which approximates max-min fairness for each queue in its end-to-end congestion management via Forward Explicit Congestion Notification [126, 4]. In the simulation experiments, we also implemented an idealized version of max-min fairness, which provides an upper bound on the performance achievable by *any* congestion-control protocol targeting max-min fairness.

Metric: Our metric of interest is speedup, defined as *the ratio of the performance of a given workload on the Saba-enabled network to the performance of the workload on the baseline system*. Throughout this section, the average speedup reports the geometric mean of the result.

Testbed: We conduct our experiments on a cluster of 32 servers. Each server runs Ubuntu 18.04 and is equipped with two 8-core Intel Xeon E5-2650v2 (Ivy Bridge) CPUs at 2.60GHz. Each CPU has 20 MB of L3 cache and two hardware threads per core, though we disable SMT in our experiments. Each server has 64 GB of system memory and a single-port 56Gb InfiniBand NIC (ConnectX-3) connected on socket 0. NICs are interconnected via a Mellanox SX6036G InfiniBand switch, which supports 9 VLs (only 8 VLs are configurable). We also use an additional server to run the *centralized* controller. At profiling time, the same server runs the profiler. In all experiments, we reserve 100% of the link capacity to be managed by Saba (i.e., C_{Saba} = 100%).

On the application side, we use the Spark and Flink frameworks. We use ten workloads from Intel's industry benchmarks [55] running on top of Spark and Flink. The workloads and the evaluated dataset sizes are summarized in Table 3.1.

Simulation: To assess Saba at a larger scale, we implement it in Mellanox's InfiniBand

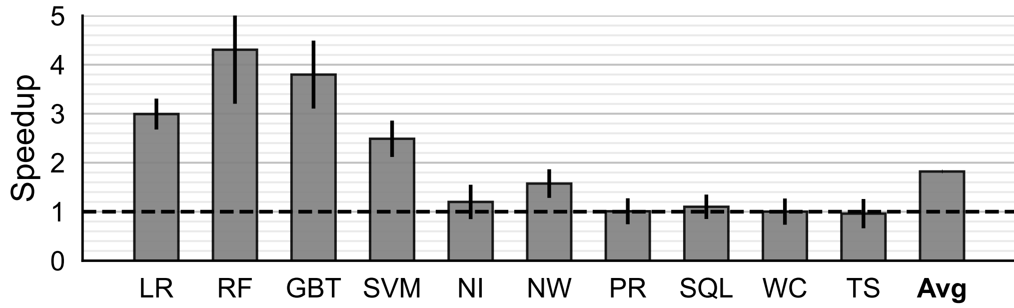


Figure 4.13: Speedup of workloads with Saba over the baseline.

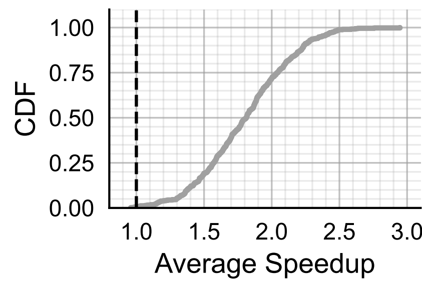


Figure 4.14: CDF of the average speedup of 500 cluster setups.

simulator based on OMNeT++ [89]. We configure the simulator to support 56Gbps link capacity per port to match our hardware testbed. Each port supports 16 VLs, each with a dedicated queue. We simulate a representative network configuration with a Spine-Leaf topology and three levels of switches [112]: 54 spine, 102 leaf, and 108 top-of-rack switches. Each top-of-rack switch connects 18 servers, for a total of 1,944 servers in the network. Similar to our testbed setup, we set C_{Saba} to 100%.

We generate 20 distinct synthetic workloads in the simulator. Each workload emulates the computation and communication stages, which is a common pattern in parallel frameworks such as Spark and Flink. The amount of computation, communication, and the number of stages varies across the workloads to emulate varying degrees of bandwidth sensitivity. In the simulation, each server runs one workload. In a topology with 1,944 servers, each of the 20 workloads has 97 instances, which are randomly distributed across the network.

4.6.2 Main results

In this section, we evaluate the performance of Saba in an environment with uneven distribution of traffic on each node, which mimics realistic scenarios in private datacenters.

To create such an environment, we generate 500 *cluster setups*. In each cluster setup, 16 jobs are randomly selected by drawing, with replacement, from the set of workloads listed in Table 3.1. All workloads are profiled in advance with the degree of polynomial of 3 as described in Section 4.2.1. The dataset size of each job is randomly selected from 0.1x, 1x, and 10x of the dataset used by the profiler. The number of instances of a job is also randomly selected from 0.5x to 4x of the number of nodes used by the profiler (8 nodes). On each server, one core is assigned to each job, and memory is equally partitioned among all workloads. Instances of jobs are randomly distributed among servers with two constraints: 1) at most one instance of a given job is assigned to a server, and 2) each server accommodates at most 16 jobs. For each cluster setup, we run all jobs together two times, once with Saba managing the bandwidth of jobs, and once with the baseline; and we measure the completion time for each job.

Figure 4.13 displays the average speedup over the baseline for each workload. As the figure shows, Saba improves the average performance across workloads by 1.88× compared to the baseline. The largest performance improvement is observed on workloads with high bandwidth sensitivity. E.g., the performance of RF is increased by 3.9×, and LR by 3.6×. For 2 (out of 10) workloads that have low sensitivity to network bandwidth (TS and PR), their performance is slightly reduced by 5% and 1% over the baseline, respectively. The reason for the performance drop is that Saba distributes the bandwidth in favor of bandwidth-sensitive workloads. Thus, while workloads that are highly sensitive to bandwidth get larger shares of network bandwidth, workloads with low sensitivity receive smaller shares. This re-distribution may result in mild performance degradation for some of the workloads, as our results show.

Figure 4.14 shows the CDF of the average speedup of 500 cluster setups. As the figure shows, the average speedup of cluster setups ranges from 0.94× to 2.99×. In only 2 (out of 500) cluster setups, Saba results in a performance slowdown compared to the baseline.

4.6.3 Sensitivity studies

In Section 4.2.2, we observed that the degree of polynomial, dataset size, and the number of nodes impact the accuracy of sensitivity models. In this section, we evaluate the impact of these parameters on the performance of Saba. In all scenarios, all workloads are profiled in advanced (Section 4.2.1). In studies 1 and 2, the degree of polynomial (k) used by the profiler is 3. We vary the degree of polynomial in study 3.

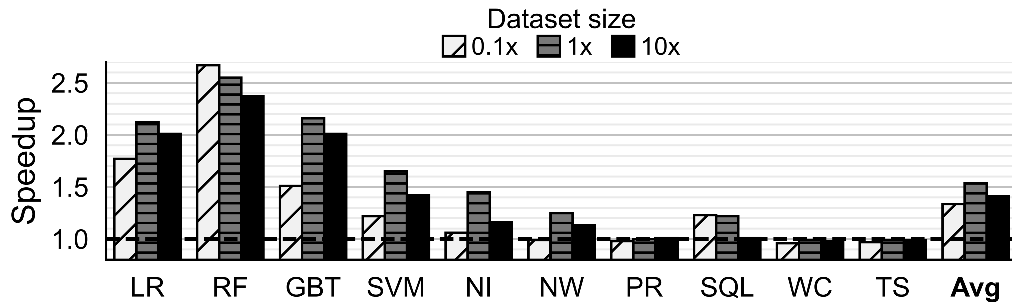


Figure 4.15: Impact of dataset size at runtime on the performance of Saba.

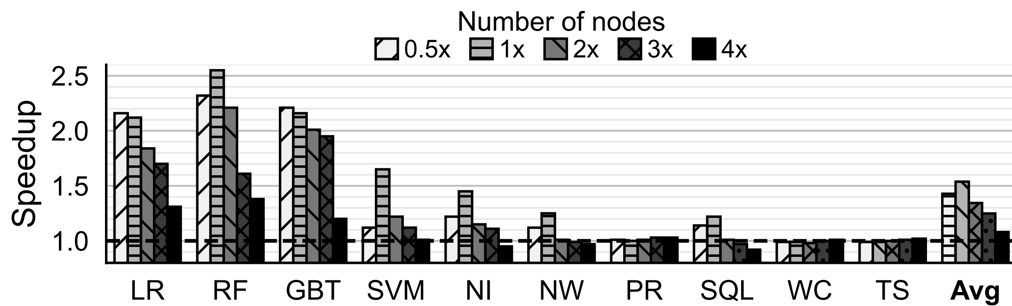


Figure 4.16: Impact of the number of nodes at runtime on the performance of Saba.

1) Impact of dataset size: As described in Section 4.2.2, the accuracy of the sensitivity models declines as the difference in dataset size at profiling and runtime increases. We compare the performance of Saba across various dataset sizes, ranging from 0.1x to 10x the size of the datasets used in the profiling phase. To this end, we create a homogeneous setup in which the number of nodes is the same number of nodes used in the profiling phase (8 nodes) and all nodes run all workloads together. We run one instance of each workload on every server with a core assigned to each workload and memory equally partitioned among all workloads.

Results are shown in Figure 4.15. As the figure shows, the applications benefit the most (54% speedup) when the runtime dataset size matches the dataset size in the profiling (1x). However, even when datasets are ten times smaller (0.1x) or larger (10x), Saba is still able to obtain performance improvement across workloads. When the workloads use the 0.1x and 10x datasets, Saba improves the average performance by 33% and 40%, respectively, compared to the baseline.

2) Impact of number of nodes: Similar to dataset size, the accuracy of a sensitivity model decreases as the number of nodes running a distributed application at runtime drifts from the number of nodes in the profiling phase (explained in Section 4.2.2). We

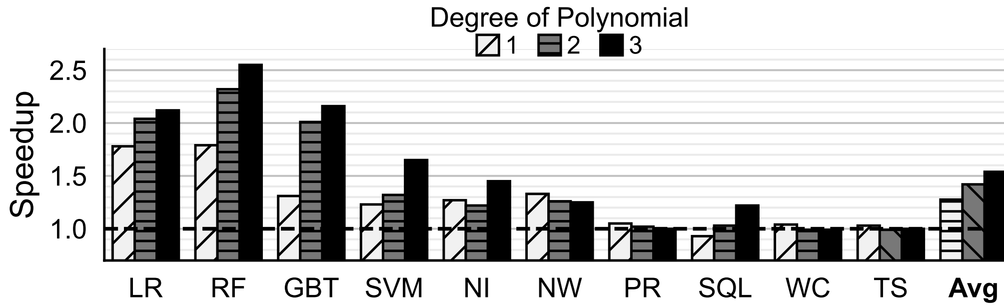


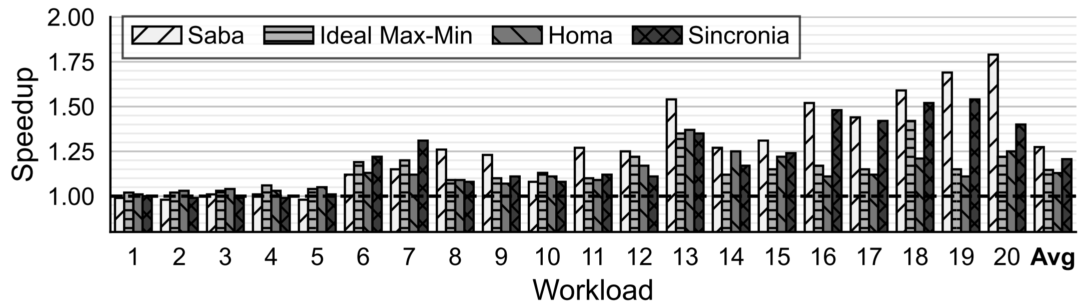
Figure 4.17: Impact of degree of polynomial on the performance of Saba.

compare the performance of Saba across a number of nodes, ranging from 0.5x to 4x of the number of nodes used by the profiler (8 nodes). Similar to study 1, we run one instance of each workload on every server. In this study, workloads use datasets of the same size as in the profiling.

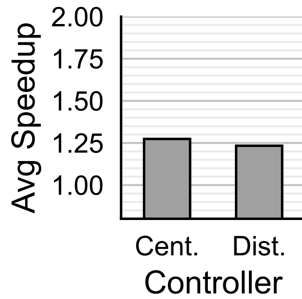
As Figure 4.16 shows, Saba achieves 42% average speedup over baseline when the number of nodes is reduced to half. Increasing the number of nodes to 2x and 3x compared to the profiling results in 34% and 26% average speedup, respectively. When increasing the number of nodes to 4x compared to profiling, we observe that Saba gains only 9% average speedup; however, workloads such as SQL, NW, and NI experience 8%, 6%, and 3% drop in their respective performance. This observation was expected as Section 4.2.2 explained that when the number of nodes at runtime is 4x, the accuracy of sensitivity models significantly drops. We conclude that when the number of nodes used in deployment exceeds the number in the profiling configuration by over 3x, the benefits of Saba significantly diminish.

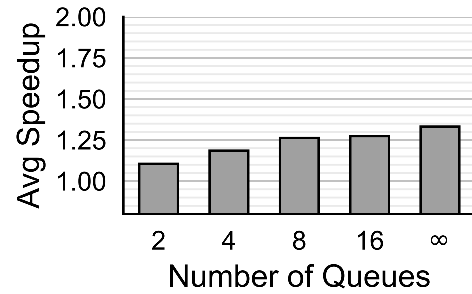
3) Impact of degree of polynomial: As explained in Section 4.2.2, the degree of polynomial plays an important role in the accuracy of the sensitivity models. We compare the performance of Saba while varying the degree of polynomial used by the profiler from 1 to 3. Similar to study 2, we run one instance of each workload on every server. In this study, workloads use datasets of the same size as in the profiling phase and the number of nodes is the same number of nodes used in the profiling phase (8 nodes).

Figure 4.17 displays the impact of the degree of polynomial on the performance of Saba. As expected, Saba benefits from more accurate sensitivity models. Some workloads benefit from the higher degrees of polynomial. For instance, when the sensitivity model of the SQL workload is generated using second- and third-degree polynomial, SQL experiences 3% and 22% improvement, respectively, over the baseline.



(a)





(b) (c)

Figure 4.18: Simulation results. (a) speedup of Saba, ideal max-min, Homa, and Sincronia, all over the baseline. (b) the average speedup with centralized versus distributed controllers. (c) impact of the number of queues on the performance of Saba.

This represents 7% performance degradation for SQL with a sensitivity model based on a first-degree polynomial. Overall, with the first- and second-degree polynomial, Saba achieves 1.27× and 1.42× average speedup, respectively.

4.6.4 Simulation results

To study the performance at a larger scale, we expand the deployment size by using the simulator and running the set of synthetic workloads as explained in Section 4.6.1.

Profiling: We calculate the bandwidth sensitivity of workloads by profiling them. For each of the simulated workloads, the profiler deploys instances of the workload on a rack-scale simulated system with 18 nodes (thus mimicking a real deployment). The profiler uses the third-degree polynomial in the following studies to generate a sensitivity model for each workload. In all studies, Saba uses a centralized controller, except for study 7.

4) Saba vs. ideal max-min fairness:

Ideal max-min fairness: In the ideal implementation of max-min fairness, each workload is assigned to a dedicated queue, and packets from queues are serviced using the Round-Robin algorithm. In this scheme, in each turn, the scheduler in the switch selects a queue and chooses the packet at the head of the queue. Assuming that all packets have the same size if such a scheme transmits one packet in each turn, it achieves the upper bound of max-min fairness [52].

We evaluate the performance of synthetic workloads at a large scale with ideal max-min fairness and compare that with Saba. We run all workloads together in the simulation. Similar to Section 4.6.2, Saba uses 8 per-port queues in the switches.

Figure 4.18a presents the results. As the figure shows, almost all workloads achieve higher performance using Saba compared to using ideal max-min fairness. The maximum speedup achieved by a workload is 79%; while the performance of two workloads is penalized by 3%. The average speedup across workloads for Saba and ideal max-min fairness is 27% and 14%, respectively. This shows that per-flow max-min fairness is inherently unable to directly target the application-level performance, as it tries to achieve bandwidth fairness at the flow level, but ignores the fact that some workloads are more sensitive than others. In contrast, Saba allocates the bandwidth of each network link based on the bandwidth sensitivity of the workloads using it. Thus, workloads with higher sensitivity get more bandwidth and see a performance improvement over max-min fair allocation.

5) Saba vs. Homa: In this study, we compare Saba against the recently-proposed Homa [94], which is considered the state-of-the-art networking protocol designed for datacenters. Similar to Saba, Homa leverages the priority queues available in network switches. Homa prioritizes short flows to achieve optimal flow-level completion time. We use Homa's OMNet++ simulator with the same topology and set of synthetic workloads described in Section 4.6.1.

As Figure 4.18a shows, Homa achieves 12% speedup over the baseline. Saba outperforms Homa by an additional 13%. The reason that bandwidth-sensitive workloads benefit from Saba more than Homa is the fact that Homa differentiates flows based on their size. E.g., in this setup, Homa assigns all flows longer than a certain size (10KB) to the *same* priority queue, without differentiating their associated workloads; thus, Homa does not allocate bandwidth in favor of bandwidth-sensitive workloads and the application-level performance of workloads is ignored. Saba, however, differentiates workloads at runtime based on their application-level sensitivity to bandwidth, resulting in improved performance.

6) Saba vs. Sincronia: To bridge the gap between flow-based bandwidth allocation schemes and application-level bandwidth requirements, a networking abstraction, called coflow, has been proposed recently [23]. Coflow represents a collection of related flows to convey application-specific communication requirements. In this study, we compare Saba against Sincronia[2], which is the state-of-the-art clairvoyant coflow scheduler. Sincronia tries to minimize the coflow completion time by ordering all unfinished coflows and assigning priority levels to the flows according to their coflow order. Sincronia requires flow sizes to be known a priori. While such a requirement is not feasible in datacenters [152], it provides Sincronia with near-optimal coflow completion time. To enforce the allocated rates, Sincronia leverages the underlying priority-enabled transport layer.

As Figure 4.18a shows, Sincronia achieves 19% speedup over the baseline. Indeed, the coflow abstraction allows workloads to more accurately express their bandwidth requirements to the network fabric; however, it does not take the sensitivity of workloads into account and the overall application-level performance of workloads is ignored. In addition, Sincronia, like other coflow-based approaches, needs applications to be modified and invoke coflow API [23]; Saba, however, requires *no* modification to applications.

7) Centralized vs. distributed: In this study, we evaluate the impact of the centralized versus distributed controller on Saba's performance. We repeat Study 4 with the distributed controller. As explained in Section 4.3.4, in Saba, with the distributed controller, the profiler performs the application-to-PLs and hierarchical clustering operations offline. Thus, the controller may not have the most accurate mappings, leading to a trade-off between performance and scalability.

As Figure 4.18b demonstrates, Saba with the distributed controller is able to achieve a 1.23× speedup over the baseline, falling just 4% short of Saba with the centralized controller. We conclude that using the distributed controller slightly reduces the effectiveness of Saba while improving scalability.

8) The impact of the number of queues: The number of queues per output port varies among switches used in today's datacenters. 8 queues are common, though more capable switches may support 16 queues per port [94]. We study the impact of the number of queues in network switches on the performance of Saba. To do so, we repeat study 4 and use switch configurations with 2, 4, 8, and 16 queues per port. We also study a configuration with unlimited queues, where each workload is assigned to a dedicated queue; this configuration provides an upper bound on Saba's performance.

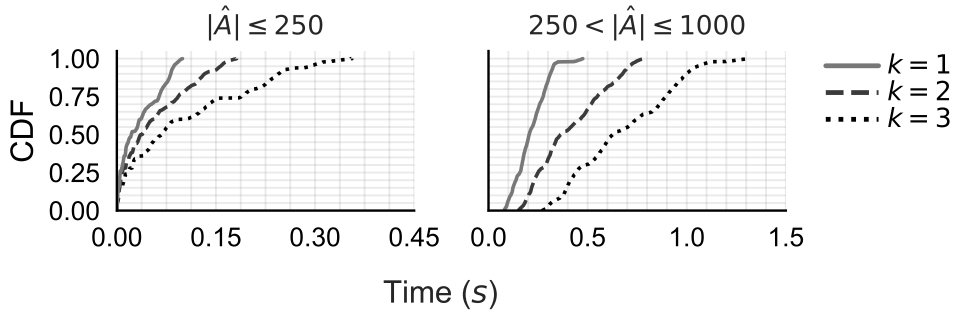


Figure 4.19: Overhead of a centralized controller.

Figure 4.18c plots the results of the study. Compared to the baseline, Saba delivers a 1.12× average performance improvement with just 2 queues at each output port. With 8 queues, Saba achieves a speedup of 1.27×, approaching the ideal speedup of 1.33× with an unlimited number of queues. This result shows that 8 queues, which is common in today's datacenter switches, is sufficient for Saba to achieve close to its optimal performance.

4.6.5 Overhead of the controller

In study 3, we observe that modeling the bandwidth sensitivity of workloads with higher degrees of polynomial results in higher performance. In this study, we investigate the added overhead by higher degrees of polynomial in bandwidth calculation on a large scale in the simulator. To this end, we evaluate the *calculation time* of a centralized controller, i.e., the time the controller takes to compute the bandwidth share of applications for all switches. We generate 30,000 scenarios, in which the size of the active application set varies from 1 to 1,000. In each scenario, 32 instances of each application are randomly distributed among nodes.

Figure 4.19 displays the CDF of measured calculation time with various sizes of active application sets ($|\hat{A}|$). The result shows that for sets of applications with sizes up to 250, the calculation time of the controller at 99$^{\text{th}}$ percentile is 0.09s, 0.16s, and 0.31s for $k = 1$, $k = 2$ and $k = 3$, respectively. By increasing the size of applications set up to 1,000, the calculation time of the controller at 99$^{\text{th}}$ is 0.43s, 0.72s and 1.13s for $k = 1$, $k = 2$ and $k = 3$, respectively. Despite the higher accuracy achieved by sensitivity models with higher degrees of polynomial, the increased overhead in bandwidth calculation reduces the responsiveness of the controller. While the degree of the polynomial must be carefully tuned according to the number of applications in the deployment of Saba,

the calculation time of the controller even with $k = 3$ is negligible as compared to the runtime of workloads. To put the calculation time in perspective, the studied workloads take from several minutes to hours to finish their job, while in an extreme case, the calculation time of a centralized controller for 1,000 applications takes 1.13 seconds. More than that, the datacenter operator can distribute the controller, or use a multi-threaded implementation of the controller and run it on multiple cores or accelerators such as FPGAs to reduce the calculation time.

4.6.6 Discussion

How does the input dataset size during profiling affect Saba's performance, and can Saba overfit in this scenario? Our result shows that the impact of the input dataset during the profiling phase is a crucial consideration in the performance improvement provided by Saba. In practice, accurately estimating the dataset size during profiling can be challenging, and there may be a desire to use a smaller dataset for profiling to reduce time and cost. This introduces a potential discrepancy between the dataset size used for profiling and the one encountered at runtime.

Regarding the possibility of overfitting, it is worth noting that Saba primarily profiles a given application using a single dataset size during its design phase. Consequently, there is a potential risk of the sensitivity model becoming overly tailored to that specific dataset size. However, the results of the study in Section 4.6.3 indicate that even when the runtime dataset size varies significantly—ranging from ten times smaller (0.1x) to ten times larger (10x) than the dataset used for profiling (1x)—Saba consistently achieves performance improvements across different workloads. This observation suggests that while overfitting to a particular dataset size is a possibility, the sensitivity models retain their predictive power effectively, demonstrating Saba's robustness.

Exploring alternative approaches, one option is to profile the application with multiple dataset sizes during the design phase. This approach could help mitigate the risk of overfitting by exposing the sensitivity model to a broader range of dataset sizes. We can take a further step by adding dataset size as an additional dimension within the model. While this could potentially enhance accuracy by accounting for various dataset sizes, it might also introduce greater complexity into the allocation algorithm. Such complexity may require advanced statistical techniques like ANCOVA (Analysis of Covariance) [35] to manage effectively. ANCOVA is particularly useful in analyzing how a continuous independent variable, in this case, bandwidth, impacts a dependent

variable like application performance while controlling for the influence of other covariates or factors such as dataset size. ANCOVA helps disentangle the specific impact of bandwidth from the noise introduced by dataset size. The implementation of ANCOVA adds a layer of complexity to the allocation algorithm (specifically eq. (4.2)) and necessitates careful consideration of the computational resources required by the controller, as datacenter operators desire to keep the overhead of allocation (Section 4.6.5) as low as possible. Decisions regarding these trade-offs would depend on specific application requirements and performance objectives.

How does Saba adapt to applications that evolve or change over time? Saba operates based on the profiling phase without real-time awareness of dataset size changes during runtime. Saba considers the characteristics of the dataset size observed during the profiling phase and optimizes resource allocation accordingly for the entire application's lifetime. It does not adapt to variations in dataset size as they occur during runtime.

However, the findings in this study indicate that the optimization strategies of Saba remain effective even when the dataset size during runtime differs significantly from the profiling phase. This underscores Saba's robustness in scenarios where applications may experience changes in their resource requirements over time, demonstrating its ability to maintain effective bandwidth allocation despite variations in the dataset size of applications.

Does Saba achieve global optimality in bandwidth allocation? If not, how could it be extended to do so? Saba's approach to bandwidth allocation revolves around finding locally optimal solutions based on its available profiling information. However, achieving global optimality in bandwidth allocation is fraught with challenges:

The effectiveness of Saba can be affected by disparities in the profiling setup, such as differences in dataset size and the number of nodes, compared to the runtime configuration. These variations can influence allocations, potentially causing deviations from globally optimal solutions. Furthermore, Saba operates without a comprehensive understanding of the application's internal structures and dependencies. This lack of awareness can result in allocations that do not align with globally optimal solutions. An extension to Saba could involve incorporating insights from application behaviors to enhance the overlap of communication and computation phases across applications and improve network resource utilization. Additionally, Saba currently makes bandwidth allocation decisions at the application's lifetime granularity. To move closer to global optimality, an extension could involve continuous real-time monitoring of application behaviors and resource usage. This approach would enable more adaptive and optimized

decisions, considering changing conditions over time.

It is important to note that achieving a globally optimized scheduling allocation poses significant computational challenges, as it falls into the category of NP-complete problems [140]. Particularly in the context of large datacenters, attaining true global optimality may be infeasible. Consequently, Saba's existing approach centers on seeking locally optimal solutions within the given constraints and available information. Potential extensions aim to further improve Saba's adaptability and efficiency in addressing varying runtime conditions.

What limitations must be considered in the adoption of Saba, and which components of the stack require porting? To facilitate the adoption of Saba, it is crucial to consider the limitations and determine which parts of the stack need to be ported. Saba can be integrated into the existing infrastructure by adding its software interface at one of the following levels within the stack:

1) Application: In order to access the full range of features offered by the Saba library, applications can directly call the interface functions. This approach is the easiest to implement from a technical standpoint, as it simply requires calling the relevant functions within the application code. However, it does require modification to the application itself in order to integrate with the Saba library. While this may require some additional work upfront, it ultimately allows applications to fully leverage the application-aware bandwidth allocation provided by Saba, leading to improved performance and efficiency.

2) Framework: An alternative approach to integrating with the Saba library that does not require modification to individual applications is to integrate Saba with frameworks, such as Spark and Flink. These frameworks are able to call the interface functions from within the framework itself, making Saba transparent to the applications. This means that only the framework needs to be modified in order to take advantage of Saba's bandwidth allocation, rather than each individual application. This can be a more efficient and convenient approach as it reduces the amount of work required to integrate with the library. However, it does require the use of a supporting framework, which may not be suitable for all use cases.

3) Transport: There is a third approach to integrating with the Saba library that provides even greater transparency compared to the previous two options. This approach involves coupling the transport layer with the connection manager, which allows all applications to be Saba-compliant without any modification. However, implementing this approach does present some technical difficulties. Specifically, the transport layer must be

configured to register all applications, as existing transport protocols do not provide any support for Saba.

4.7 Summary

Saba uses bandwidth sensitivity of workloads to drive bandwidth allocation, resulting in a bandwidth assignment that is well-correlated with workloads' actual bandwidth needs. Our evaluation reveals that in our real-world setup, Saba improves the average performance across a wide range of workloads by 1.88× compared to InfiniBand. While bandwidth-sensitive workloads benefit the most from Saba, workloads with low sensitivity face little or no performance degradation with Saba.

Furthermore, we evaluate Saba at scale in simulation using synthetic workloads and compare it against InfiniBand, an ideal implementation of max-min fairness and Homa. Our evaluation shows that Saba improves the average performance across studied workloads by 1.27×, 1.11× and 1.13× compared to InfiniBand, ideal implementation of max-min fairness and Homa, respectively.

Finally, we evaluate the impact of the number of queues per port at switches on the performance of Saba. We observe that with just two queues, Saba effectively outperforms InfiniBand in terms of the performance of workloads running on them. Moreover, existing switches that support eight per-port queues are capable to get close to the performance of the ideal implementation of Saba.

Chapter 5

Characterization of an InfiniBand Switch

Datacenters feature an ever-growing mix of traditional and emerging applications that place high-performance demands on the datacenter network. As explained in Chapter 3, some of these applications, such as big-data analytics using Hadoop or Spark [123, 142, 29], distributed machine-learning training [139, 46, 113, 80, 147, 10], data backup and VM migrations, require exchanging large amounts of data among nodes, necessitating high bandwidth. These types of applications may require different strategies for bandwidth allocation in order to meet their specific performance needs.

Other applications, such as those relying on disaggregated memory [37, 120, 67, 49, 3, 121] and distributed in-memory storage [36, 83, 82, 87, 19, 32, 63, 92, 39], require ultra-low network latency to provide the illusion of a scale-up system. In many cases achieving the lowest possible per-packet latency (on the order of a few microseconds) is critical to the success of the service, making tail latency (e.g., 99.9^{th} percentile) an important metric to consider [37, 27].

To meet the network latency and bandwidth needs, datacenter operators [15] have begun deploying high-end networking solutions in the form of InfiniBand [14]. Initially developed for the HPC domain, these networks tend to combine custom fully offloaded network stacks, RDMA capability, and lossless links to provide high end-to-end performance. A number of recent works have demonstrated that, indeed, InfiniBand-based deployments can offer low latency (order of microseconds) and high bandwidth for a given application, such as a distributed in-memory KVS [32, 40, 61, 68]. In most of these cases, network parameters are tuned for an individual application to harness the maximum potential of an InfiniBand fabric [144]. In practice, however, in public and

private datacenters, multiple applications with different latency and bandwidth demands might coexist in a cluster and share the network [149].

In this chapter, by focusing on latency-sensitive applications, we aim to answer the following question:

How well do the existing InfiniBand switches support the resulting mix of latency-sensitive and bandwidth-intensive traffic?

To answer this question, we study a rack-scale InfiniBand deployment with a single ToR switch, representing the simplest fieldable cluster setup. For the purpose of evaluating a switch in such a rack configuration, it is imperative that the measurement methodology be accurate, particularly one that is capable of measuring the latency of a switch under stress in isolation (that is, without endpoint processing overhead). We observe that existing performance measurement tools for RDMA-based networks suffer from endpoint processing overheads (local and remote) which affects the accuracy of latency measurements.

As a solution to this problem, we developed RPerf, a tool for measuring RDMA performance at a high level of precision. RPerf overcomes deficiencies of existing tools to precisely measure latency and avoids the need for expensive hardware-based solutions or support for hardware timestamping on the NICs.

Using RPerf, we observe that our InfiniBand setup achieves very low latency in an unloaded network, corroborating prior work. Further, we are able to consistently utilize high bandwidth as the number of bandwidth-intensive flows is varied. Our InfiniBand switch, however, cannot provide low latency for a latency-sensitive flow in the presence of bandwidth-intensive flows. In order to achieve low latency without compromising throughput, we examine several strategies, such as varying packet sizes or using priority levels, but all of the evaluated approaches are found to be insufficient. Using an InfiniBand switch simulator, we also examine different packet scheduling policies within the switch and observe that readily available packet scheduling policies, such as First Come First Serve and Round-Robin, are unable to guarantee performance isolation for both latency-sensitive and bandwidth-intensive flows.

Based on our findings, we conclude that the contemporary InfiniBand gear (NICs and the switch) used in our evaluation may be effective for applications with homogeneous traffic, but is unable to accommodate the heterogeneous demands of modern datacenters.

In short, the main contributions of this chapter are as follows:

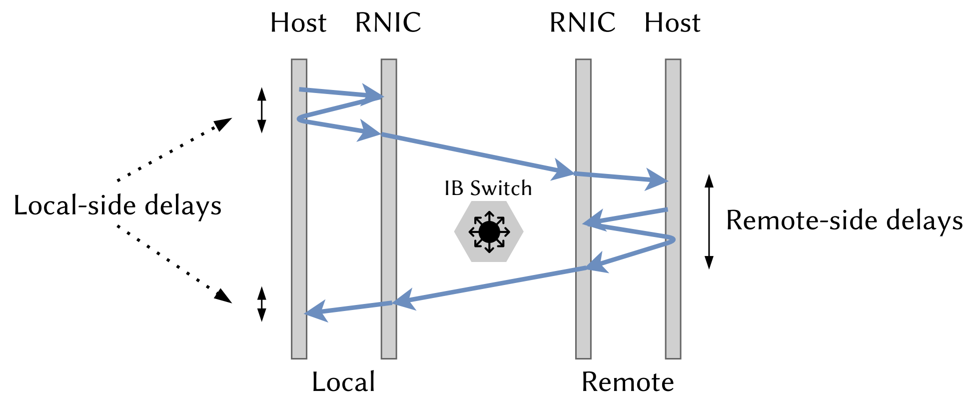


Figure 5.1: Ping-pong style RTT calculation.

- We introduce RPerf (Section 5.2), an accurate micro-benchmarking tool, which provides sub-microsecond precision measurement. We compare RPerf against existing measurement tools (Section 5.3).

- To improve the performance of co-existing applications, we examine several strategies, including the use of different packet sizes and priority levels (as described in Section 5.4). We show each of these strategies has its own limitations and may not be sufficient on its own and discuss the trade-offs and shortcomings of each approach in order to better understand the challenges and limitations of optimizing the network for co-existing applications.

5.1 InfiniBand latency measurement

Thanks to fully offloaded network stack processing and other high-performance features summarized in Section 2.3, InfiniBand achieves sub-10 microsecond latency in an unloaded network. Such low latency presents several challenges for accurate NIC-to-NIC latency measurement.

The main challenge is isolating the latency of the switch from other components, particularly the software and PCIe. An ideal solution is to directly measure one-way port-to-port latency through the switch; doing so, however, requires the use of expensive data acquisition devices [151]. Another option is to use precise sub-microsecond clock synchronization at the endpoints [108]; however, this approach assumes that one-way latencies in both directions are the same, which is not the case when traffic is congested, particularly with a converged traffic pattern.

An alternative approach for latency measurement is a ping-pong style test to find

the round-trip time (RTT) in software (Figure 5.1). Problematically, RTT calculation can be biased by *remote-side processing*, which is required to generate and transmit the response packet at the remote end. Such remote-side processing includes the software overhead for generating the response and the PCIe transactions necessary for transferring data to/from the RNIC. This remote-side processing delay does not reflect true network latency and, as such, should be excluded from the measurement. In addition, RTT calculation suffers from *local-side processing* delays. Local-side processing includes multiple PCIe transactions that the local RNIC performs to fetch data from the host memory, putting the data into packets and enqueuing packets. As software captures the posting time of a request, not transmitting time, the calculated RTT includes PCIe transactions, RNIC processing, and queuing delays, and results in biased measurement.

Existing latency measurement tools

Several methodologies and tools are available for measuring the performance of datacenter networks [145, 108, 51, 41]. Some of these tools leverage hardware timestamping, a feature now becoming more prevalent in commodity NICs. One such tool is MoonGen [33], a highly adaptable and high-speed packet generator built upon the Intel Data Plane Development Kit (DPDK) [31]. MoonGen stands out by offering the capability to execute user-provided Lua scripts on a per-packet basis, enabling it to deliver precise and granular latency measurements. Another example is Lancet [69], a tool for precise latency measurement. Lancet is designed to measure the open-loop tail latency of microsecond-scale datacenter applications with high fan-in connection patterns. The unique design of Lancet is that it can identify situations where tail latency measurements may be inaccurate due to factors like workload configuration, client infrastructure, or application intricacies. Similar to MoonGen, Lancet leverages NIC-based hardware timestamping for robust end-to-end RPC latency measurements when available. In cases where hardware capabilities are lacking, Lancet uses an asymmetric setup with a latency agent to reduce client bias, making it a versatile tool for accurate latency assessment. Unfortunately, none of the above tools are designed to precisely measure the latency of an InfiniBand switch, particularly under load.

There are only a few RDMA-based tools available for measuring the latency over an InfiniBand fabric. RDMA Bench [62] benchmarks an RDMA-based fabric from the application layer and does not isolate the NIC-to-NIC latency. Perftest [99] consists of

a collection of micro-benchmarks that use a ping-pong latency measurement approach. In Perftest, the remote side responds to a ping with a pong generated in software; consequently, it suffers from the remote-side processing problem. QPerf [100], another micro-benchmarking tool, calculates the RTT at a high load using a *post-poll* measurement approach. In the post-poll approach, a QPerf client posts a write request and then polls for the completion of the request; therefore, the QPerf server does not respond to the request in software. Such an approach removes the remote-side software overhead; however, it still includes the PCIe delays for DMA-ing the data into remote memory, which is required for a write request (Figure 2.2b). QPerf also fails to perform precise tail latency measurement (it does not track per packet latency) and only reports the average latency. The accuracy of both Perftest and QPerf is further diminished because both tools include local-side processing delays in their measurements. To conclude, existing measurement tools fail to factor out local-side and/or remote-side processing overheads, which impedes their ability to accurately measure the latency through the switch.

5.2 RPerf

Existing methodologies for estimating latency in InfiniBand switches are unreliable or produce inaccurate results. This is a significant issue as latency is an important factor in the performance of a network and any errors in measurement can lead to flawed optimization or design decisions. To address this problem, we introduce *RPerf*, a micro-benchmarking tool specifically designed to accurately measure the latency of InfiniBand switches under various load conditions. RPerf uses a novel approach that allows it to provide precise and reliable latency estimates, making it a valuable tool for network engineers and researchers.

RPerf is designed to measure the RTT between local and remote hosts. It leverages RDMA verbs to accurately measure the latency of InfiniBand switches without including the delays at the endpoints. In the following sections, we will describe key aspects of the design of RPerf.

5.2.1 Excluding remote-side processing

In order to exclude all software overheads at the remote end, RPerf adopts the post-poll approach, in which only the local host posts a request and polls for completion. By

leveraging the RC transport, in which the remote RNIC generates a response without involving the destination host, RPerf avoids software processing overheads at the remote end. To exclude the PCIe latency on the remote end, RPerf uses the RDMA send verb. RDMA send causes the remote RNIC to generate a response to the source RNIC immediately upon the receipt of the request and without waiting for the PCIe transaction to complete at the remote end (see Figure 2.2d). The combination of using post-poll and RDMA send avoids biasing latency measurements with remote-side software overheads and PCIe delays.

5.2.2 Excluding local-side processing

When the application posts a request using one of the InfiniBand verbs, the RNIC handles the request asynchronously and the control returns to the application. Meanwhile, the host sends the request to the local RNIC through PCIe and the local RNIC performs a DMA read to fetch the data for the RDMA send operation (Figure 2.2d). After fetching data, the local RNIC processes, enqueues, and eventually transmits the request. The sum of latencies incurred by these actions at the local host and the RNIC make up the *local-side processing overhead*.

In order to avoid including local-side processing delays in RTT, RPerf calculates local-side processing overhead for every RDMA send request so that it can be excluded from the measurement of the switch latency. To do so, RPerf leverages *loopback* messages, which are messages that are sent from a host to itself via the local RNIC. Specifically, after sending an RDMA send to the destination host (which we call an over-the-wire send), RPerf immediately generates a *loopback RDMA send* request at the local host and times it. The latency of the loopback request is the local-side processing overhead, which can then be subtracted from the latency of the over-the-wire send.

5.2.3 RTT calculation

Using the ideas introduced above, we now describe how RPerf precisely measures the RTT through an InfiniBand switch. The process for RTT measurement is shown in Figure 5.2. At the outset, the local host posts a pair of RDMA send requests: an over-the-wire request and a loopback, storing the posting time (T_P). While the over-the-wire request is being sent out, the loopback work request is processed by the local RNIC, which generates a CQE when it is finished; RPerf captures the completion time for the loopback request (T_L). When the ACK for the over-the-wire request arrives at the

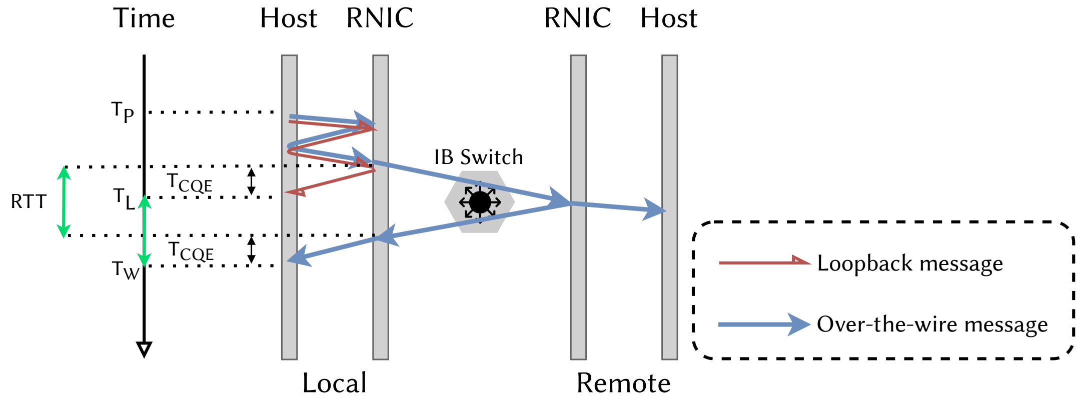


Figure 5.2: RTT calculation by RPerf.

local RNIC, the RNIC issues a CQE. RPerf records this completion time as (T_W) and calculates the RTT as follows:

$$RTT = (T_W - T_P) - (T_L - T_P) = T_W - T_L \quad (1)$$

RPerf subtracts the time it takes for a loopback message to be completed in order to eliminate the time taken for the over-the-wire request to be processed at the local RNIC and the local PCIe latency. This allows RPerf to effectively identify and remove these delays, accurately measuring the latency of the InfiniBand switch.

Additional details: In order to minimize software-induced performance variability, each RPerf thread is pinned to a CPU core and Huge Pages are allocated for all required buffers. For capturing the timestamps of events accurately, RPerf uses Time Stamp Counter (TSC) through *rdtsc* x86 assembly instruction, which offers high-accuracy timestamping measurement within user space. RPerf follows Intel recommendations for TSC calibration and access [101]. Multiple instances of RPerf can be run on different servers, and a user can specify a traffic pattern (e.g., one-to-one or many-to-one) to measure specific aspects of the system, such as zero-load latency, peak bandwidth, or latency at load.

5.3 Evaluation of InfiniBand switches

In this section, we study the performance of an InfiniBand switch on (1) a 7-server InfiniBand testbed with a suite of workloads and (2) a simulated 7-server cluster with a set of synthetic workloads.

Table 5.1: Bandwidth-intensive workloads.

Workload	Dataset Size
Logistic Regression	10k samples
SVM	150k samples
NWeight	Graph size (# of edges): 4250M
PageRank	50M pages
Join (SQL)	Two tables (# of records): 5000M and 120M

5.3.1 Methodology

Hardware testbed: In our experiments, we use seven identical hosts with dual-socket Intel Xeon E5-2630 v4 (Broadwell) CPUs running at 2.20GHz and 64GB of RAM. The hosts run Ubuntu server 18.04 LTS with kernel version 4.15.0-50 and are equipped with InfiniBand Mellanox MT27700 ConnectX-4 RNICs [88]. These RNICs are connected through a Mellanox SX6012 InfiniBand switch with 12 QSFP ports, 16 MB of buffer capacity per port, and 9 VLs [90]. The switch and RNICs have a peak bandwidth of 56Gbps and Mellanox reports port-to-port latency of up to 200ns through the switch.
Simulator: We use a modified version of the InfiniBand OMNeT++ simulator, developed by Mellanox. This simulator models a network with seven nodes connected to an InfiniBand switch and provides two different packet scheduling policies: First Come, First Served (FCFS) and Round-Robin (RR). The parameters of the modeled switch, including peak bandwidth, port-to-port latency, and number of VLs, are set to match those of our actual InfiniBand switch.
Workloads: To study sharing characteristics of the InfiniBand switch under traffic from applications with different objectives, we consider two major types of applications:

1) Latency-Sensitive (LS) Application: In our study, a latency-sensitive workload is exemplified by the use of RPerf, our microbenchmark. RPerf operates by sending RC packets synchronously in a closed-loop fashion between a pair of nodes (source and destination). The message size varies in different experiments. Within our hardware testbed, each RPerf instance computes the RTT using the methodology detailed in Section 5.2.

2) Bandwidth-Intensive (BI) Application: We use five workloads from Intel's industry benchmarks [55] running on top of Apache Spark [141]. The workloads and the evaluated dataset sizes are summarized in Table 5.1. We use Mellanox's SparkRDMA plugin [91] to enable InfiniBand data transfers in Spark. This plugin transfers data using

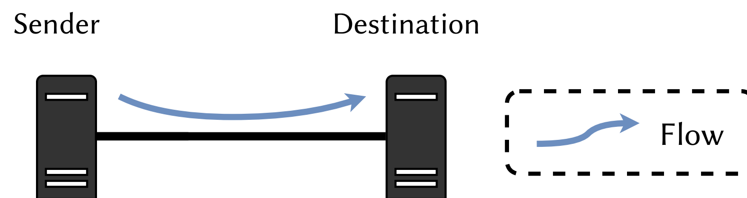


Figure 5.3: Back-to-back setup, where two servers are directly connected.

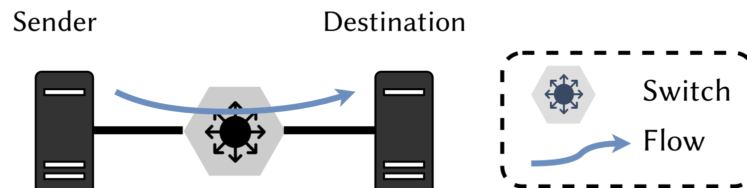


Figure 5.4: One-to-one setup, where two servers are connected through a switch.

RC packets asynchronously (open-loop). The message size and the number of instances vary in different experiments. RPerf calculates the bandwidth of BI applications during the tests.

Metrics: We consider the 99.9th latency percentile as the tail. In all experiments, we run the test three times and the duration of each test is 15 minutes. All graphs plot the average values of the three runs; we do not plot standard errors, as they are negligible (below 0.001).

5.3.2 Performance under one-to-one traffic

In this section, we study the performance of the switch in a one-to-one setup, whereby applications run a pair of servers. First, we evaluate the isolated latency of the switch using RPerf by measuring the RTT with and without the switch. Then, we measure the end-to-end RTT using Perftest and Qperf. Finally, we evaluate the peak bandwidth of applications with and without the switch.

5.3.2.1 Latency of LS traffic

Without switch: We first measure the RTT without the switch by directly connecting the RNICs of a generator and the destination server (Figure 5.3). A server sends LS messages to the destination server in the first test to determine RTT at zero-load.

Figure 5.5 shows the RTT distributions of two hosts directly connected for different payload sizes measured by RPerf. As the figure illustrates, the median RTT for 64B

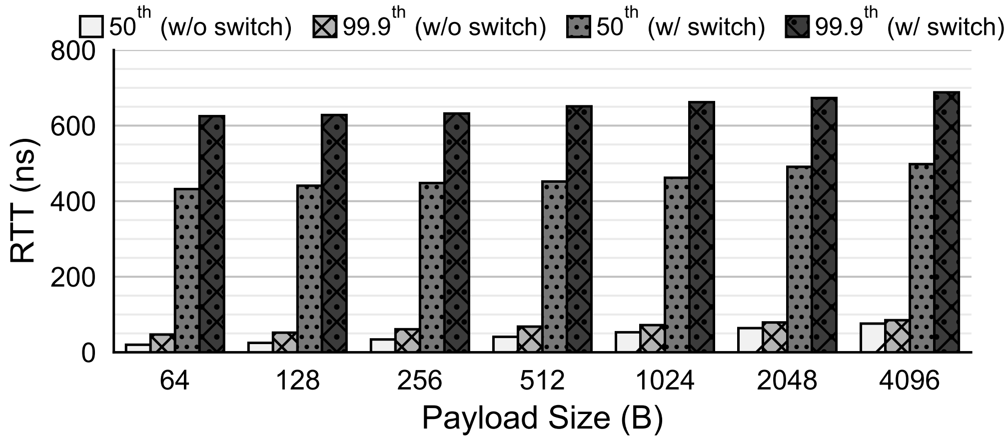


Figure 5.5: RTT calculated by RPerf for different packet sizes with and without the switch.

is 20ns, and the tail RTT is 47ns. When the message size is increased to 4096B, the median and tail RTT grow to 76ns and 85ns, respectively. The difference between the median and tail RTT of a setup without the switch is at most 30ns. These results demonstrate that the RTT is low and the effect of payload size on RTT is small.

With switch: Next, we study the performance of the InfiniBand switch. We connect two servers (one sender, one destination) through the switch (Figure 5.4) and run the same one-to-one traffic pattern as above. We examine the RTT at zero-load for messages sent by the generator.

As Figure 5.5 shows, the median RTT for 64B messages is 432ns and the tail is 625ns. Moreover, increasing the payload size to 4096B results in the median and tail RTT of 498ns and 688ns, respectively.

Ground truth: The study demonstrates RPerf's accuracy by comparing its reported latencies. Without the switch, RPerf reports median latencies between 20 and 85ns, and with the switch, latency measurements fall between 432 and 498ns. These measurements closely align with the reported latency by the switch manufacturer, Mellanox, which stands at 400ns (200ns one-way port-to-port latency, i.e., 400ns RTT [90]). The negligible differences observed between RPerf and Mellanox's reports, ranging from 32 to 98ns, can be attributed to the transmission delay of packets, as they are very close to the reported latency without a switch (between 20 and 85ns).

The assessment of RPerf's accuracy through hardware-based solutions is a crucial aspect of evaluating its performance measurement capabilities. However, as of the current study, the in-depth examination of RPerf's accuracy using hardware-based validation remains a subject for future research, meaning that while the study has

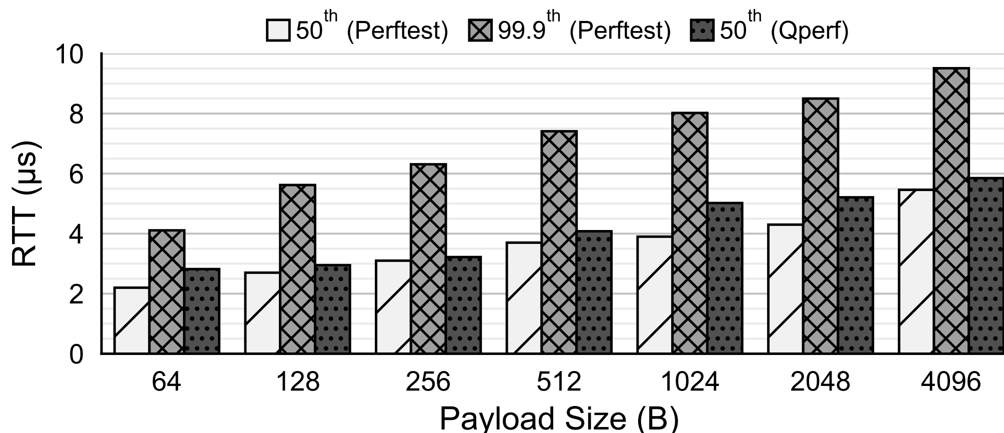


Figure 5.6: End-to-end RTT calculated by Perftest and Qperf for different packet sizes with the switch.

demonstrated the tool's accuracy through comparisons and measurements, a more comprehensive validation using dedicated hardware setups and benchmarks is left as a direction for future work. We conclude that RPerf serves as a reliable benchmark for assessing and validating the latency performance of RDMA-based switches, offering a ground truth measurement that closely aligns with manufacturer specifications.

Tail latency: Figure 5.5 shows that regardless of the size of the payload, the difference between the median and tail RTT through the switch is $\approx$ 200ns. By comparing to the no-switch setup, in which the difference between the median and tail RTT is at most 30ns, we can deduce that the switch introduces at least a 170ns delay to the tail RTT, which is about 45% of the median RTT; therefore, the switch suffers from tail latency, even at zero-load.

Using existing tools: Finally, we measure the end-to-end RTT with the switch using Perftest and Qperf.

Figure 5.6 shows the RTT distributions of two hosts connected through the switch calculated by Perftest and Qperf for different payload sizes. In Figure 5.6, Perftest reports 2.20μs median RTT and 4.11μs tail RTT for 64B. Furthermore, by increasing the message size to 4096B, the median and tail RTT grow to 5.46μs and 9.51μs, respectively. Figure 5.6 also shows that the median RTT reported by Qperf is 2.82μs for 64B. By increasing the message size to 4096B, the median grows to 5.85μs. Unfortunately, Qperf does not report tail RTT.

We find that although Perftest and Qperf are useful for measuring end-to-end latency, their calculated latency is significantly (an order of magnitude) higher than the reported

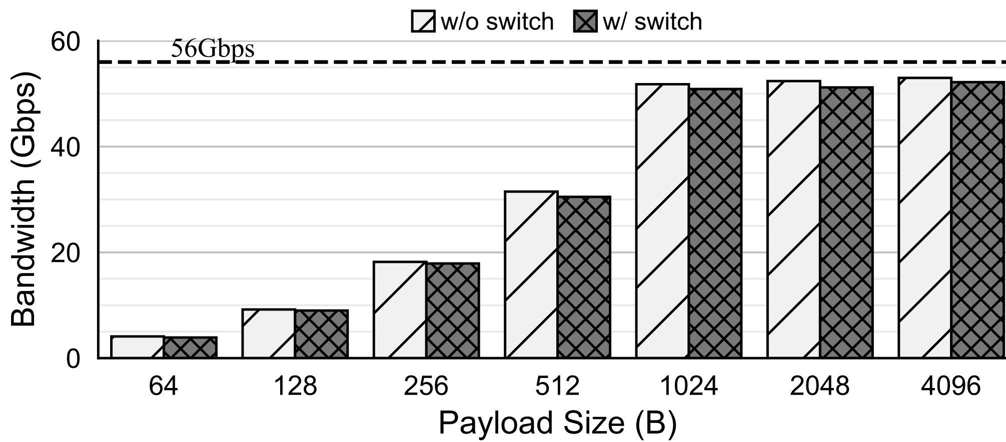


Figure 5.7: Bandwidth for different packet sizes with and without the switch.

latency in the switch specification. This observation suggests that these tools are unable to isolate the switch latency due to endpoint overheads, especially when the message size is increased. We conclude that Perftest and Qperf are not suitable for precisely measuring the latency of the InfiniBand switch.

Take-aways:

1. Without the switch, the latency between a pair of RNICs connected back-to-back is extremely low, well under 100ns for all evaluated payload sizes.

2. RPerf is able to exclude almost all the endpoint delays, which existing InfiniBand latency measurement tools fail to do.

3. Using RPerf, the median RTT through the switch is 432-498ns, depending on the payload size. This latency is close to the expected 400ns round-trip latency per the switch spec.

4. The switch increases the tail latency to about 45% above the median latency.

5. Existing tools are unable to isolate the switch latency due to endpoint overheads and flawed methodology.

5.3.2.2 Bandwidth of BI traffic

This section evaluates the maximum achievable bandwidth for a BI workload with different payload sizes. In this study, one instance of Logistic Regression workload runs on each server.

Without switch: We first measure the bandwidth without the switch by directly connecting the RNICs of two servers (the same setup as Figure 5.3).

As Figure 5.7 shows, the attained bandwidth varies with payload size. Using a payload size in the range of 1024B to 4096B, the workload can achieve 51.8 to 53Gbps at the destination port, showing that with large payload sizes, our setup attains over 90% of the bandwidth of a 56Gbps link. However, the achieved bandwidth is very poor with small payloads; e.g., with 64B messages, the bandwidth is 4.1Gbps, meaning that less than 10% of link capacity is utilized. This problem is largely due to two reasons:

1) Header size of an InfiniBand packet can be up to 52B[14]; hence, less than 56% of the frame is the payload for a 64B message.

2) To achieve line rate bandwidth, here 56Gbps, the RNIC must be capable of processing $\approx$ 110 million 64B packet per second, which is beyond the RNIC's capability; this problem is well known and [62] discusses the reasons.

With switch: Next, we study the maximum achievable bandwidth of the InfiniBand switch. We connect two servers through the switch (Figure 5.4) and run the Logistic Regression workload on them as above.

Figure 5.7 illustrates the bandwidth achieved for different payload sizes over the switch. With payload sizes of 64B and 4096B, our setup attains 3.9 and 52.2Gbps, respectively. We can observe that the bandwidth achieved through the switch is slightly lower (by up to 0.8Gbps) than without the switch.

Take-aways:

1. While over 90% of link capacity can be achieved with large packet sizes, bandwidth utilization is poor with small packets.

2. The switch has a negligible effect on the bandwidth of the BI application in the one-to-one setup.

5.3.3 Coexistence of flows with different types

In this section, we evaluate the network performance of the InfiniBand switch in the presence of mixed-type flows (Figure 5.8). In this setup, a varied number of instances of Logistic Regression workload (from one to five) send BI flows with 4096B payload size to one destination server, forming a converged traffic pattern[1]. Meanwhile, an

[1] While the Logistic Regression job forms an all-to-all communication model, here we only focus on the flows that are sent to the 'destination' server.

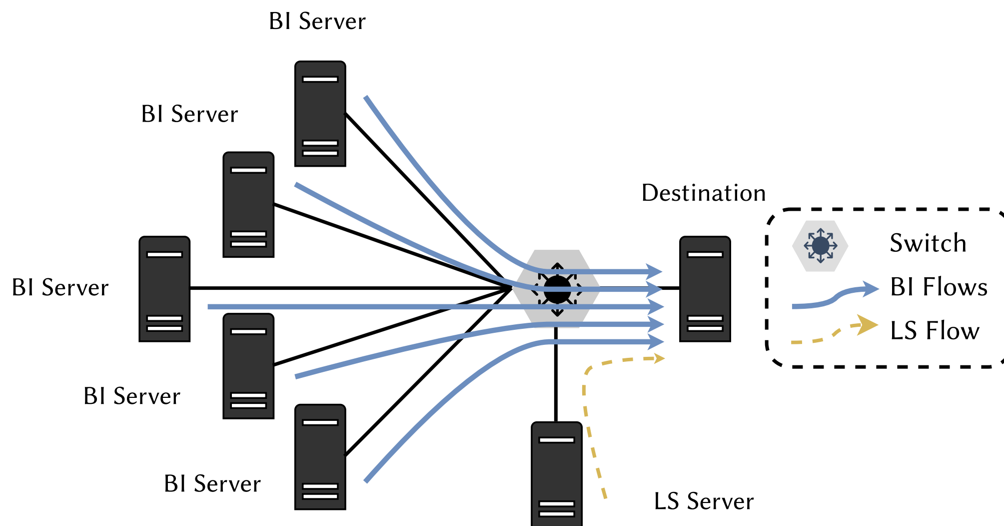


Figure 5.8: Mixed-flow-type setup, where from 1 to 5 servers asynchronously send BI flows (number of servers varies in different runs) and 1 server sends LS flow, all to the same destination.

LS workload sends latency-sensitive flows with 64B messages to the same destination server.

Bandwidth-intensive flows hurt latency-sensitive flows: Figure 5.9 shows the median and tail RTT of the LS flow as we vary the number of instances of Logistic Regression workload. With one active BI flow, the median and tail RTT of LS flow through the switch is 0.6μs and 0.9μs, respectively. Adding a second BI flow increases the median and tail RTT of LS flow to 5.2μs and 5.7μs, respectively. The third BI flow further worsens the latency of LS flow by increasing the median and tail RTT to 10.7μs and 12.6μs, respectively. As the figure shows, adding yet more BI flows further degrades the median and tail latency of the LS flow. Indeed, with each added BI flow, the median RTT of the LS flow increases by 4.8μs to 6.1μs, leading us to conclude that the switch fails to provide latency isolation in the presence of bandwidth-intensive flows, and that latency-sensitive flows are *unprotected*.

Bandwidth-intensive flows receive equal share: Figure 5.10 illustrates the total bandwidth achieved by BI flows as we vary the number of instances of Logistic Regression workload. As Figure 5.10 shows, the bandwidth attained by one active BI flow is 52.2Gbps. With two BI flows, each flow achieves 25.5 to 25.6Gbps, resulting in an overall bandwidth of 51.1Gbps. When five BI flows are active, the bandwidth per BI flow ranges from 9.3 to 9.9Gbps, with overall attained bandwidth of 48.4Gbps. While one would not expect the total bandwidth through the switch to vary as a function of

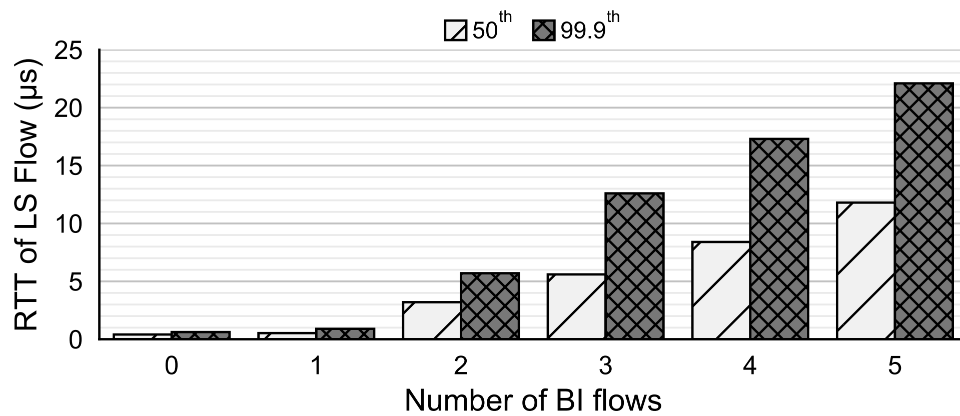


Figure 5.9: RTT of LS flow

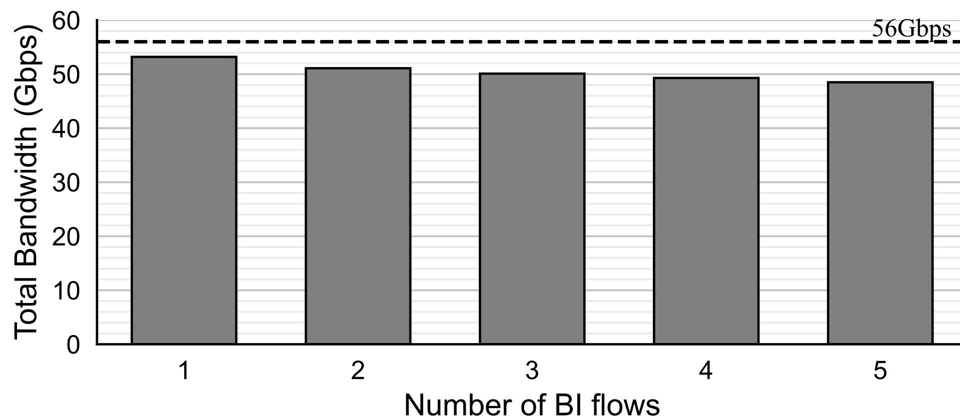


Figure 5.10: Total bandwidth of all BI flows

the number of active flows, we observe that increasing the number of BI flows from one to five deteriorates the total achieved bandwidth of all flows by 7% (from 52.2 to 48.4Gbps).

Take-aways:

1. The latency observed by the latency-sensitive source is proportional to the number of active bandwidth-intensive flows, indicating that the switch fails to provide latency isolation.

2. Increasing the number of convergent bandwidth-hungry flows diminishes the total achieved bandwidth through the switch.

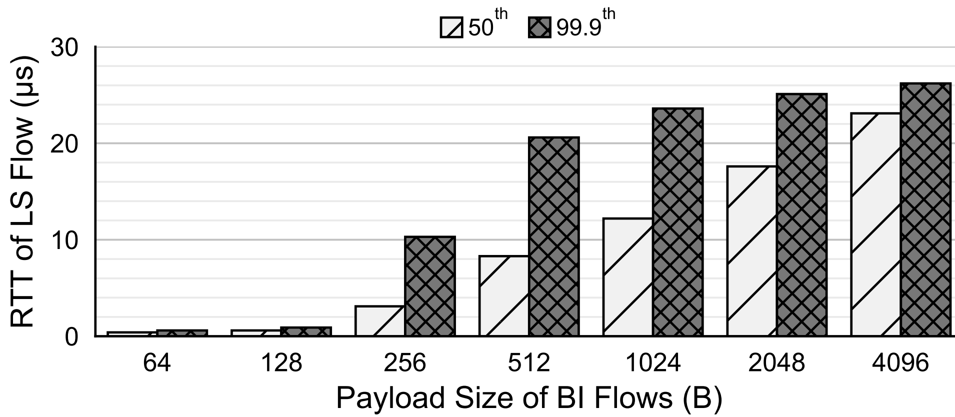


Figure 5.11: RTT of the LS flow. Note that BI flows have different message sizes in each test.

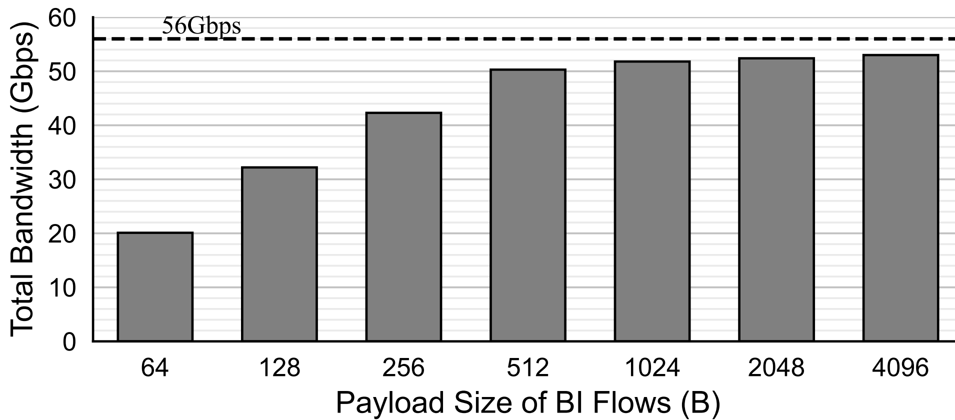


Figure 5.12: Total bandwidth achieved by BI flows as a function of the message size.

5.4 Attempts to protect latency-sensitive flows

In Section 5.3.3, we observe that the InfiniBand switch fails to isolate latency-sensitive flows from bandwidth-intensive ones. In this section, we explore different approaches to help the switch in providing protection for latency-sensitive flows.

5.4.1 Bandwidth-intensive flows with different message sizes

Our observation of high latency for LS flows in the presence of BI flows leads us to hypothesize that small messages suffer from *head-of-line blocking*; in other words, InfiniBand cannot preempt a larger payload from BI flows immediately for a shorter one from the LS flow and forces LS packets to wait for a long time while the large

messages are transmitted. Thus, we attempt to mitigate the head-of-line blocking caused by bandwidth-intensive flows by reducing the payload size of such flows. To do so, we conduct an experiment to see whether a smaller payload size for BI flows could facilitate rapid preemption and improve LS latency without sacrificing bandwidth for BI flows.

We direct five instances of Logistic Regression workload sending flows to one destination server. The payload size of BI flows varies in different tests. We also use *batching* with small payload sizes to improve bandwidth utilization. At the same time, one server sends LS flow with 64B messages to the same destination server.

Figure 5.11 shows the RTT of the LS flow in the presence of flows from Logistic Regression workload (with different payload sizes in different tests), and Figure 5.12 shows the overall bandwidth that the BI flows can achieve with different payload size.

According to Figure 5.11, small payload sizes for BI flows lead to low LS latency. For instance, when Logistic Regression uses a payload size of 64B, the median and tail RTT of LS flow are 0.4μs and 0.6μs, and with 128B payload size of BI flows, the median and tail RTT of LS are 0.6μs and 0.9μs. However, using small payloads for the BI flows sacrifices their ability to achieve high throughput. With 64B or 128B payload sizes, the BI flows can barely utilize 35% or 70%, respectively, of link capacity at the destination port.

Meanwhile, as Figure 5.12 shows, if Logistic Regression generates flows with large payload sizes, they can achieve high bandwidth utilization. Using a payload size in the range of 512B to 4096B, BSGs can achieve from 88% to 93% of link capacity at the destination port. However, choosing a large payload size for BI flows hurts the latency of the LS flow. With a 512B payload size, the median and tail RTT of the LS flows are 20.0μs and 20.6μs. Larger payloads further worsen the median and tail RTT of the LS flow (26.3μs and 28.2μs for 4096B).

Take-away: By reducing the payload size of bandwidth-intensive flows, we can achieve *either* low-latency for LS flows *or* high bandwidth for the BI flows, but not both at the same time.

5.4.2 Packet scheduling policy at the switch

In Section 5.4.1, we observe that reducing the payload size of bandwidth-intensive flows does not resolve the latency-bandwidth trade-off. In this section, we take another step in an attempt to prevent latency-sensitive flows from being stalled by bandwidth-intensive

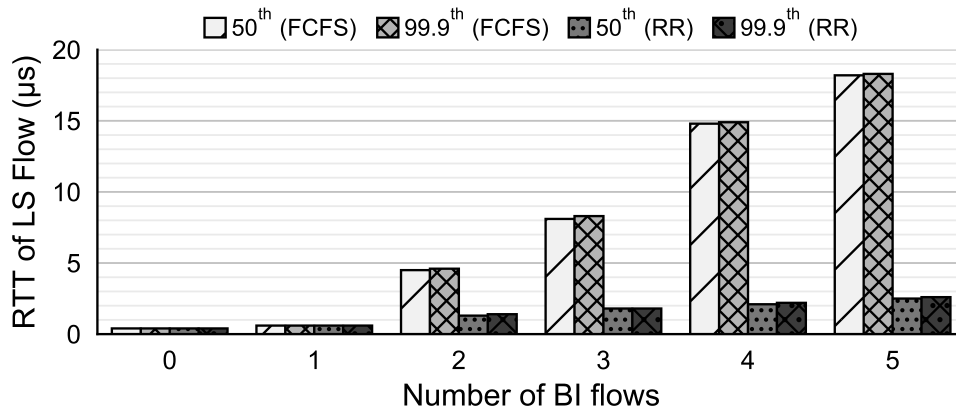


Figure 5.13: The impact of the number of BI flows on the RTT of LS flow in the simulator.

ones. To this end, we study the impact of different in-switch packet scheduling policies on per-flow latency and bandwidth.

We consider a policy to be fair if the time each flow spends in the switch is proportional to the size of the flow. Thus, if the switch uses an unfair policy for packet scheduling, latency-sensitive flows might be stalled by bandwidth-intensive ones. Such a policy is unfair because it does not take flow size into account and fails to perform proportionally fair scheduling at each turn.

As the scheduling policy of our switch is not configurable, we use the InfiniBand simulator (Section 5.3.1) to assess the effect of scheduling policies on fairness. We use the same setup as in Section 5.3.3, with five servers asynchronously sending 4096B flows and one server sending LS flow with 64B messages to the same destination server. The InfiniBand simulator provides two different packet scheduling policies: First Come, First Served (FCFS), and Round-Robin (RR). We calculate the RTT of LS flow in the converged setup using different packet scheduling policies and compare them with the real switch. Note that Mellanox documentation does not specify the scheduling policy implemented in our switch, so one of our goals is to understand the implemented policy.

FCFS policy: Figure 5.13 shows the median and tail RTT of the LS flow as we vary the number of the BI flows in the simulator under the FCFS scheduling policy. In the absence of a BI flow, both the median and tail RTT of LS are 0.4μs. With one active BI flow, both the median and tail RTT of the LS flow are 0.6μs. Adding the second BI flow increases the median and tail RTT of the LS flow to 4.5μs and 4.6μs, respectively. With five active BI flows, the median and tail RTT are 18.2μs and 18.3μs, respectively. We can observe that the median and tail RTT in the simulator are almost identical (0.1μs difference). Thus, unlike the real switch, the simulator does not introduce significant

tail RTT. The reason is that the switch uArch is not modeled in detail in the simulator; therefore the median and tail RTT of the simulator are much closer, compared to the median and tail RTT of the real switch. In the simulator, each additional BI flow adds a delay of 3.9μs to 4.6μs to the median RTT of the LS flow. This trend closely matches the behavior observed with the real switch, where each additional BI flow adds 4.6μs to 5.2μs to the LS flow's latency.

To investigate the additional delay added by each BI flow, we look into the architecture of the simulated switch. In the modeled switch, each input port has dedicated buffering used for absorbing bursts. With the FCFS policy, in each turn, the arbiter examines the packet at the head of each input buffer and chooses the oldest packet. In our converged traffic experiment, each BI flow fills to capacity its respective input buffer. Once an LS packet enters the switch, it too is enqueued at its port's input buffer. Following the FCFS policy, the arbiter selects the LS packet only after *all* other packets present at the switch when the LS packet arrived have been scheduled. Therefore, in our setup, the minimum amount of time an LS packet needs to wait can be computed as follows:

$$W_t = \frac{N \times BufferSize}{LinkBandwidth} \tag{2}$$

where N is the number of ports BI flows occupy (i.e., whose input buffers are full), BufferSize is the size of each input buffer, and LinkBandwidth is the bandwidth of a link.

In the modeled switch, the size of each buffer is 32KB, and the link bandwidth is 56Gpbs. In this case, each additional BI flow adds 3.6μs waiting time to each LS packet, which is close to the latency observed in both the simulator and the real switch.

Take-away: When using the FCFS policy, the simulator exhibits latency unfairness and behaves much like a real switch.

RR policy: With RR policy, the arbiter in each turn selects a port and chooses the packet at the head of the port. In this case, whenever an LS packet arrives, it waits for at most the number of active ports. Having such a policy facilitates rapid preemption and mitigates head-of-line blocking.

Figure 5.13 shows the median and tail RTT of LS flow when different numbers of BI flows are active in the simulator with the RR scheduling policy. Without any active BI flow, both the median and tail RTT of the LS flow are 0.4μs. With one active BI flow, both the median and tail RTT of the LS flow are 0.6μs. By increasing the number of active BI flows to five, the median and tail RTT grow to 2.5μs and 2.6μs.

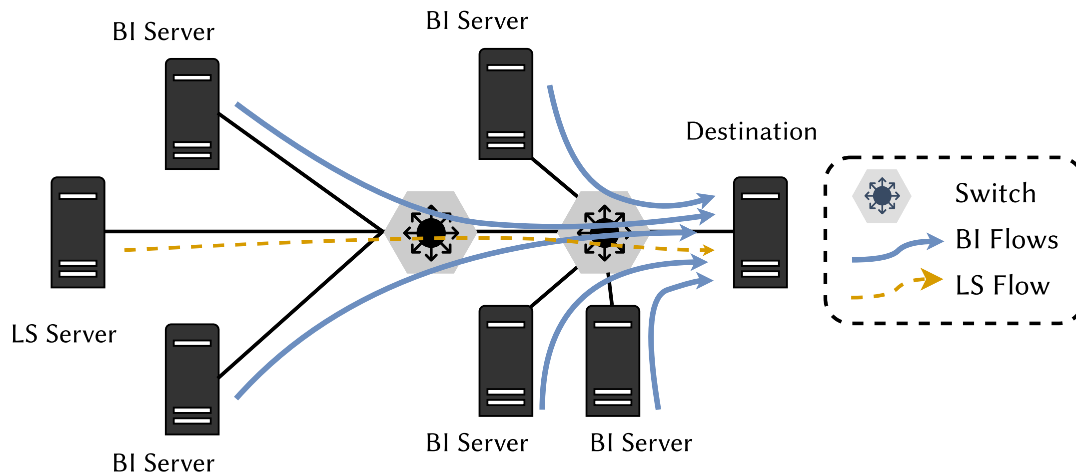


Figure 5.14: Simulation with a multi-hop setup.

Unlike the experiment with the FCFS policy, with the RR policy, increasing the number of BI flows does not change the RTT for LS dramatically and the latency of latency-sensitive messages is well controlled.

Take-aways:

1. The measurements attained on the simulator with the RR policy are vastly different from those on the real switch. This further indicates that the real switch uses the FCFS scheduling policy.

2. Unlike the FCFS policy, the RR scheduling policy is more effective at protecting the latency-sensitive flow.

Packet scheduling policies in a multi-hop topology: At first glance, it seems that the RR policy on the switch resolves the dilemma of isolation and protection of a latency-sensitive flow. However, can the RR policy continue to be effective in a multi-hop topology? To answer this question, we extend our simulated setup to a two-hop topology, where a pair of switches are connected together (Figure 5.14). Two instances of the BI workload and one LS workload are connected to the upstream switch, and three instances of the BI workload are attached to the downstream switch. The destination server is also attached to the downstream switch. All BI instances send 4096B messages to the destination server.

We calculate the RTT of the LS messages in the multi-hop setup using different packet scheduling policies and compare them with each other. In each test, the packet policy of *both* switches is either FCFS or RR.

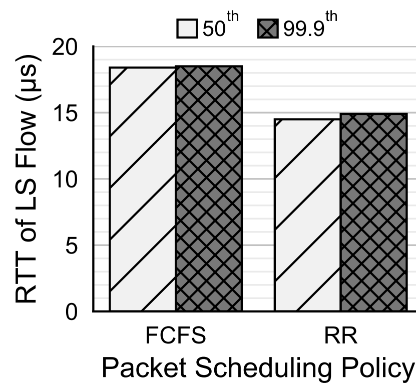


Figure 5.15: RTT of the LS flow in a multi-hop setup.

Figure 5.15 shows the median and tail RTT of the LS flow when different packet policy is used in the switches in the simulator. Using the FCFS policy in both switches, the median and tail RTT of the LS flow are 18.4μs and 18.5μs. Using the RR policy, the median and tail RTT of the LS flow are 14.5μs and 14.9μs.

We can observe that if a latency-sensitive flow shares a link (in this setup the link that connects two switches) with bandwidth-intensive flows, the RR policy is unable to protect the latency-sensitive flows. The reason is that the latency-sensitive flow will be queued at the same input buffer as the bandwidth-intensive flow in the downstream switch, and will hence suffer from head-of-line blocking.

Take-away: The RR policy fails to isolate latency-sensitive flows in a multi-hop setup.

5.4.3 Queue separation through priority levels

Previous experiments highlight the importance of allocating separate buffer resources and scheduling priorities for latency-sensitive and bandwidth-intensive flows in order to optimize performance. One of the key observations from these experiments is that indeed, head-of-line blocking, which occurs when flows contend for the same queue, serves as the root cause of increased latency in latency-sensitive applications. To address this issue, we delve into the concept of queue separation as a potent solution. By separating different types of flows and assigning them different priorities, it is possible to ensure that latency-sensitive flows are not blocked by bandwidth-intensive flows. This can be achieved through the use of quality-of-service (QoS) configuration, such as that provided by the InfiniBand fabric. InfiniBand QoS allows for the differentiation of flows and separation of queues through the use of Service Levels (SLs) and Virtual Lanes

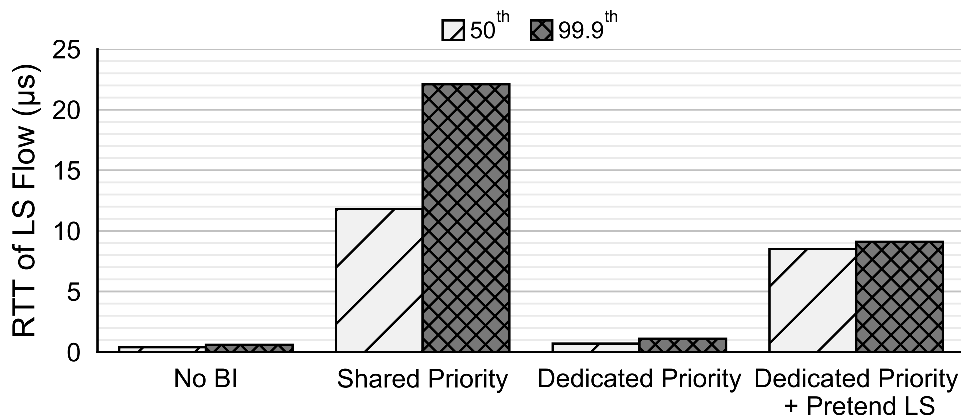


Figure 5.16: RTT of the real LS flow in different setups.

(VLs). By leveraging these capabilities, we can effectively manage the allocation of resources and prioritize different types of flows as needed, ensuring that the performance of latency-sensitive applications is not compromised by competing bandwidth-intensive traffic.

To ensure that latency-sensitive flows are not negatively impacted by bandwidth-intensive flows, we can assign a dedicated SL to latency-sensitive flows at the local host, and map this SL to a high-priority VL in the switch. This allows us to prioritize latency-sensitive traffic and ensure that it is given priority over bandwidth-intensive flows when accessing shared resources, such as buffers and queues. It is important to note that the concept of latency sensitivity is not directly recognized in InfiniBand terminology. As such, we choose to define latency-sensitive flows as those involving small messages (up to 256B). This allows us to differentiate between latency-sensitive and bandwidth-intensive flows and apply the appropriate QoS configuration to each, ensuring that the performance of latency-sensitive applications is not compromised.

The following experiment evaluates the effectiveness of using dedicated priority levels for latency-sensitive and bandwidth-intensive traffic, by assigning SL0 to the BI flows and SL1 to the LS flows. In the switch, SL0 is mapped to low-priority VL0, and SL1 is assigned to high-priority VL1.

Figure 5.16 shows the median and tail RTT for the LS traffic using a dedicated priority level. As the figure shows, using a dedicated priority protects the latency-sensitive flows. While with the shared priority (the same as Section 5.3.3) the median and tail RTT of the LS flow is 20.2μs and 22.1μs, with a dedicated priority level, LS packets have 0.7μs median and 1.1μs tail RTT. The figure shows that using a dedicated

priority level improves the latency of LS by $\approx 29\times$ for the median and $\approx 20\times$ for the tail RTT. Compared to the shared priority setup, the RTT of the LS flow with a dedicated priority level is closer to the RTT of the LS flow in the absence of BI flows (0.4μs and 0.6μs). Moreover, the total bandwidth achieved by five BI flows is the same as that achieved without using a dedicated priority level (Section 5.3.3), which indicates that such flow differentiation does not introduce a throughput penalty.

Note that queue separation can be an effective solution for safeguarding latency-sensitive applications in multi-hop switch setups too. This effectiveness arises from the fundamental principle that, even in a multi-hop configuration, latency-sensitive and bandwidth-intensive flows do not share the same queue for buffering. Unlike the single-queue scenario where head-of-line blocking can occur, the separation of queues in the multi-hop setup ensures that latency-sensitive flows are isolated from bandwidth-intensive ones. Consequently, head-of-line blocking, which hampers the performance of latency-sensitive applications by causing delays, is averted. The segregated queues maintain the necessary prioritization and resource allocation at switches, enabling latency-sensitive applications to experience minimal disruptions even in multi-hop switch topologies.

Take-away: Queue separation through differentiating flow types and assigning priority to each type can effectively protect latency-sensitive flows.

Is using priority queues a solution to the latency-bandwidth trade-off? While assigning a dedicated priority to small messages may seem promising, it is possible that a bandwidth-intensive flow could abuse (intentionally or not) this approach by pretending to be a latency-sensitive flow in order to receive more bandwidth than it would otherwise be allocated. This scenario could result in an unfair allocation of bandwidth, as the bandwidth-intensive flow would be able to receive more than its fair share. To game the QoS, a BI workload can send large amounts of data segmented into small packets. To show how this BI flow harms the bandwidth of other BI flows, we devise a test with a dedicated priority for latency-sensitive flows, in which a BI flow pretending to be an LS flow (referred to as a *pretend LS*) sends 256B messages asynchronously. We run all five Spark workloads (described in Table 5.1) together on five servers in the cluster. To do so, we run one instance of each BI workload on one server, called the destination server, with two cores assigned to each workload and memory equally partitioned among all workloads. We also run one instance of each BI workload on a server. In one test, Logistic Regression pretends to be an LS workload and transfers data with small (256B) payloads. Meanwhile, a real LS workload from

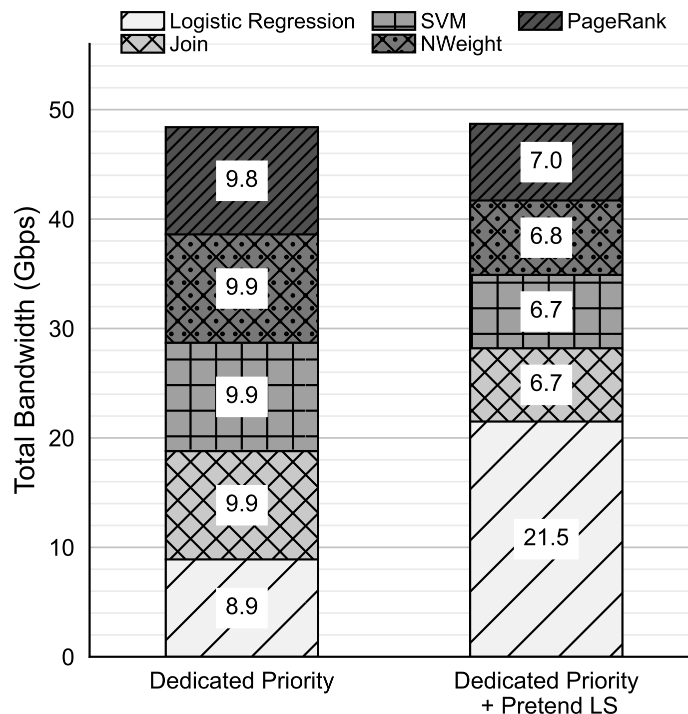


Figure 5.17: Total bandwidth achieved by BI workloads under converged traffic.

the seventh server sends latency-sensitive flows to the same destination server, forming a converged traffic pattern with BI workloads.

Figure 5.16 shows the median and tail RTT of LS traffic, and Figure 5.17 illustrates the total bandwidth achieved by all BI flows and the pretend LS (when all flows are active), along with their share of the bandwidth. Figure 5.16 shows that Logistic Regression, i.e., the pretend LS, hurts the latency of the real LS flow (8.5μs median and 9.1μs tail RTT), as both pretend and real LS flows have the same priority, and hence, the same queue on the switch. Moreover, Figure 5.17 illustrates that Logistic Regression achieves 21.5Gbps bandwidth, while each of the other BI workloads achieves 6.7 to 7Gbps. We can observe that a bandwidth-intensive source can pretend to be latency-sensitive and take three times higher bandwidth share compared to other bandwidth-intensive sources, leading to bandwidth unfairness. One might think that limiting the bandwidth for each SL/VL mapping will prevent gaming the priority-based setup; however, imposing such a limit will hurt the latency of the LS, especially when a burst of latency-sensitive packets arrives at a switch.

Take-away: By differentiating flows using priority levels, latency-sensitive flows can be protected in an environment with different types of flows. The risk, however, is that it opens up the possibility of gaming to achieve a higher bandwidth share by

bandwidth-intensive flows.

5.4.4 Discussion

Our evaluation reveals that the tested InfiniBand switch can either provide low latency to a latency-sensitive flow or high bandwidth for bandwidth-intensive flow(s), but not both simultaneously. We show that our switch uses the FCFS scheduling policy that is particularly harmful to latency-sensitive flows in the presence of bandwidth-intensive flows. Using a Round-Robin policy instead of FCFS can improve fairness in a single-hop topology. However, with just two network hops, latency-sensitive packets can still be blocked by other packets, leading to poor latency performance.

Motivated by these observations, we examined InfiniBand's QoS mechanism, which assigns different priority levels and separated queues to different flows. Our analysis shows that separating queues is necessary for latency-sensitive flows to achieve low latency without negatively impacting the link utilization of bandwidth-intensive flows. However, it is also possible for a bandwidth-intensive flow to abuse this system by pretending to be a latency-sensitive flow and sending small messages in bursts in order to receive more bandwidth than other bandwidth-intensive flows that do not attempt to manipulate the system. Therefore, relying solely on payload size for queue management is insufficient, and additional controls are necessary.

How can these findings be applied to Ethernet-based RDMA switches? In light of our findings, the implications extend to RoCE (RDMA over Converged Ethernet) switches, which, like InfiniBand switches, grapple with the challenge of balancing low latency for latency-sensitive flows and high bandwidth for bandwidth-intensive flows. RoCE switches also employ similar QoS mechanisms to InfiniBand switches. RoCE switches typically provide Weighted Round-Robin (WRR) scheduling for Traffic Classes (TCs) per egress port, akin to the assignment of VLs in InfiniBand.

The insights derived from our study underscore the relevance of revisiting scheduling policies and prioritization mechanisms in RoCE switches. Comparing policies like FCFS and Round-Robin remains pertinent, as these choices significantly influence fairness and latency performance, regardless of the underlying link-level technology. By adopting Round-Robin or similar policies to distribute resources more equitably, RoCE switches can address the challenges faced by latency-sensitive flows and bandwidth-intensive flows alike.

Furthermore, the exploration of InfiniBand's QoS mechanism offers valuable lessons

for RoCE switches. Implementing queue separation strategies that assign different priority levels and separate queues for various flows can enable RoCE switches to optimize the coexistence of latency-sensitive and bandwidth-intensive flows without compromising performance.

However, the potential for abuse of these mechanisms can still exist in the context of RoCE switches similar to InfiniBand, as highlighted in our analysis. To mitigate this risk, RoCE switches may need to incorporate additional controls beyond payload size to ensure fair and efficient resource allocation.

5.5 Summary

In this chapter, we identify shortcomings in existing RDMA-based performance measurement tools and show why they are unable to accurately assess the latency of an InfiniBand switch. We introduce the RPerf performance measurement tool that leverages RDMA verbs to exclude endpoint overheads and provide a highly accurate latency measurement for RDMA-based switches. Using the precise measurements enabled by RPerf, we analyze the latency and bandwidth of an InfiniBand switch in one-to-one and many-to-one traffic scenarios. We show that the switch fails to protect the latency-sensitive flows from bandwidth-intensive ones and that the latency is proportional to the number of active bandwidth-hungry flows. We consider several strategies for improving latency fairness, including using small packet sizes for bandwidth-intensive flows and the use of InfiniBand's QoS mechanism, but find all evaluated approaches deficient in some respect. We thus conclude that better mechanisms are needed to provide performance isolation in a mixed-traffic environment.

Chapter 6

Conclusions and Future Work

Datacenters are facilities that house large numbers of servers and other computing infrastructure, and are used to store, process, and manage large amounts of data. These facilities are crucial to the operation of many businesses, organizations, and online services, as they provide the computing power and storage capacity needed to run applications, store and retrieve data, and host online services.

Datacenter networks play an important role by connecting the various components of a datacenter and enabling communication and data transfer between them. These networks typically consist of a combination of switches, routers, and other networking equipment, and are designed to handle large amounts of traffic and data transfer at high speeds.

Bandwidth allocation is an important aspect of datacenter networking, as it determines how much of the available bandwidth is allocated to different applications and services. Proper bandwidth allocation can help ensure that critical applications and services have sufficient resources to operate effectively, while also ensuring that non-critical applications do not consume too much bandwidth and negatively impact the performance of the datacenter.

Existing bandwidth allocation schemes do not take into account the specific requirements and characteristics of different applications, leading to inefficient use of resources and poor performance. This observation motivates the design of a new bandwidth allocation scheme with application-level performance in mind.

In this final chapter, we first summarize the main contributions of this thesis, we then discuss limitations.

6.1 Summary of contributions

Introducing the notion of bandwidth sensitivity

In Chapter 3, we demonstrate the shortcomings in bandwidth allocation disciplines that are based on max-min fairness or shortest-flow first on a per-flow basis. We show that such allocation schemes do not effectively utilize the network in shared environments like datacenters, as they are unable to identify the bandwidth demands of applications. Our findings show that for the most effective use of available bandwidth, the allocation scheme should be able to distinguish between applications that require a lot of bandwidth for acceptable performance and those that can function with less bandwidth without greatly extending their completion time. To this end, we introduce the notion of bandwidth sensitivity as a guiding principle to allocate bandwidth among applications and show how this metric can be learned through profiling.

Designing Saba, an application-aware bandwidth allocation scheme

In Chapter 4, we introduce Saba, an application-aware bandwidth allocation scheme, which determines the sensitivity of applications to bandwidth, and allocates bandwidth to applications according to their bandwidth sensitivity. Saba uses a combination of ahead-of-time application profiling to identify the bandwidth sensitivity of applications and runtime bandwidth allocation using lightweight software support, all without requiring any changes to network hardware or protocols. This allows Saba to effectively allocate bandwidth and improve the performance of sensitive applications without the need for complex and costly network modifications In our evaluation, Saba effectively improves the performance of co-located workloads compared to existing and ideal implementations of max-min fairness, as well as state-of-the-art SRPT-based bandwidth allocation schemes.

Characterization of latency and bandwidth of InfiniBand switches

In Chapter 5, we identify shortcomings in existing RDMA-based performance measurement tools and show why they are unable to accurately assess the latency of an InfiniBand switch. We introduce the RPerf performance measurement tool that leverages RDMA verbs to exclude endpoint overheads and provide a highly accurate latency measurement for RDMA-based switches. Using the precise measurements enabled by RPerf, we analyze the latency and bandwidth of an InfiniBand switch in one-to-one and many-to-one traffic scenarios. We show that the switch fails to protect the latency-sensitive flows from bandwidth-intensive ones and that the latency is proportional to the number of active bandwidth-hungry flows. We consider several strategies for improving

latency fairness, including using small packet sizes for bandwidth-intensive flows and the use of InfiniBand's QoS mechanism, but find all evaluated approaches deficient in some respect. We thus conclude that better mechanisms are needed to provide performance isolation in a mixed-traffic environment.

6.2 Limitations and Future Work

In this section, we highlight the limitations of our bandwidth allocation scheme and describe potential future directions for research in bandwidth allocation schemes.

Extending Saba to manage latency-sensitive applications

Given the emergence of latency-sensitive applications, such as those relying on disaggregated memory and distributed in-memory storage, ultra-low latencies are becoming a top priority in datacenters. Our observations in Chapter 5 show that queue separation is necessary for protecting latency-sensitive flows in the presence of congestion. As explained in Chapter 4, Saba is designed for bandwidth-intensive applications and assumes that a portion of bandwidth is allocated to non-Saba-compliant applications, including latency-critical applications. The flows from these applications pass through dedicated queues that are out of Saba's control.

Recent studies (e.g., [110]) have already begun characterizing the performance of applications as a function of network latency. It is possible to extend Saba to leverage these characterizations in order to manage latency-sensitive flows as well. One possible approach could be to introduce an application-level measurable metric for sensitivity to latency, model the latency sensitivity of applications through profiling (similar to Section 4.2), and perform queue assignment accordingly. By considering the specific latency requirements of different applications, Saba can better allocate resources and improve the performance of latency-sensitive applications.

Removing the need for ahead-of-time profiling

As explained in Chapter 3, the sensitivity of applications to bandwidth can be measured through extensive offline profiling. While this approach may be practical in private datacenters where applications are recurring, it is generally not possible to profile applications in public datacenters. Additionally, while we have designed Saba to minimize the resources required for profiling, ahead-of-time profiling is still an extra step for launching new applications and it would be beneficial if datacenter operators could avoid it.

To eliminate the need for ahead-of-time profiling, it is possible to extend Saba to

monitor running applications and estimate their bandwidth demands at runtime. There are several possible approaches for estimating bandwidth sensitivity at runtime. For instance, a framework-specific solution can use information about the computation and communication stages of a submitted job to estimate how much of the completion time will be spent on communication. By understanding the communication needs of a job, the solution can more effectively allocate bandwidth and improve the performance of the job.

Leveraging the sensitivity information in job allocation

Chapter 4 demonstrates how Saba leverages bandwidth sensitivity information in its approach to bandwidth allocation, which is just one aspect of resource management in datacenters. However, other resource management components, such as job allocation and VM placement, can also benefit from using sensitivity information. By using sensitivity modeling to understand the requirements and characteristics of different applications and jobs, VM placement systems can make more informed decisions about how to allocate resources and reduce network contention between co-running applications. This can help improve the overall performance of the datacenter and ensure that allocated servers are able to meet the bandwidth requirements of submitted jobs, allowing applications and services to operate at their optimal levels.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th Symposium on Operating System Design and Implementation (OSDI)*, pages 265–283, 2016.

[2] Saksham Agarwal, Shijin Rajakrishnan, Akshay Narayan, Rachit Agarwal, David Shmoys, and Amin Vahdat. Sincronia: Near-optimal network design for coflows. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 16–29, New York, NY, USA, 2018. Association for Computing Machinery.

[3] Marcos K. Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, Rajesh Venkatasubramanian, and Michael Wei. Remote memory in the age of fast networks. *SoCC 2017 - Proceedings of the 2017 Symposium on Cloud Computing*, pages 121–127, 2017.

[4] Fatma Alali, Fabrice Mizero, Malathi Veeraraghavan, and John M. Dennis. A measurement study of congestion in an infiniband network. In *2017 Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–9, 2017.

[5] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. CherryPick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 469–482, Boston, MA, March 2017. USENIX Association.

[6] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center TCP (DCTCP). *Computer Communication Review*, 40(4):63–74, 2010.

[7] Mohammad Alizadeh, Albert G. Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 63–74, 2010.

[8] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less is more: Trading a little bandwidth for {Ultra-Low} latency in the data center. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 253–266, 2012.

[9] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pFabric: minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference*, pages 435–446, 2013.

[10] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E. Dahl, and Geoffrey E. Hinton. Large scale distributed neural network training through online distillation. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

[11] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Róbert Ormándi, George E. Dahl, and Geoffrey E. Hinton. Large scale distributed neural network training through online distillation. In *ICLR (Poster)*, 2018.

[12] Apache. Flink.

[13] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. Spark SQL: Relational Data Processing in Spark. In *SIGMOD Conference*, pages 1383–1394, 2015.

[14] InfiniBand Trade Association. InfiniBand architecture specification volume 1 and 2. Technical report, 2015.

[15] Microsoft Azure. Introducing the new HB and HC Azure VM sizes for HPC, 2018.

[16] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Antony I. T. Rowstron. Towards predictable datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, pages 242–253, 2011.

[17] Hitesh Ballani, Keon Jang, Thomas Karagiannis, Changhoon Kim, Dinan Gunawardena, and Greg O'Shea. Chatty Tenants and the Cloud Network Sharing Problem. In *Proceedings of the 10th Symposium on Networked Systems Design and Implementation (NSDI)*, pages 171–184, 2013.

[18] Hitesh Ballani, Keon Jang, Thomas Karagiannis, Changhoon Kim, Dinan Gunawardena, and Greg O'Shea. Chatty tenants and the cloud network sharing problem. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 171–184, 2013.

[19] Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, and Erfan Zamanian. The End of slow networks: It's time for a redesign. *Proceedings of the VLDB Endowment*, 9(7):528–539, 2016.

[20] Adrian M. Caulfield, Eric S. Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. A cloud-scale acceleration architecture. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 7:1–7:13, 2016.

[21] Li Chen, Kai Chen, Wei Bai, and Mohammad Alizadeh. Scheduling Mix-flows in Commodity Datacenters with Karuna. In *Proceedings of the ACM SIGCOMM 2016 Conference*, pages 174–187, 2016.

[22] Mosharaf Chowdhury. *Coflow: A Networking Abstraction for Distributed Data-Parallel Applications*. PhD thesis, University of California, Berkeley, 2015.

[23] Mosharaf Chowdhury and Ion Stoica. Coflow: a networking abstraction for cluster applications. In *Proceedings of The 11st ACM Workshop on Hot Topics in Networks (HotNets-XI)*, pages 31–36, 2012.

[24] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I Jordan, and Ion Stoica. Managing data transfers in computer clusters with orchestra. *ACM SIGCOMM computer communication review*, 41(4):98–109, 2011.

[25] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient coflow scheduling with Varys. In *Proceedings of the ACM SIGCOMM 2014 Conference*, pages 443–454, 2014.

[26] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):1–22, 2013.

[27] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[28] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI)*, pages 137–150, 2004.

[29] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI 2004 - 6th Symposium on Operating Systems Design and Implementation*, pages 137–149, San Francisco, CA, 2004.

[30] Alan J. Demers, Srinivasan Keshav, and Scott Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *Proceedings of the ACM SIGCOMM 1989 Conference*, pages 1–12, 1989.

[31] DPDK. Data plane development kit.

[32] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. FaRM: Fast remote memory. *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014*, pages 401–414, 2014.

[33] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. MoonGen: A scriptable high-speed packet generator. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, volume 2015-Octob, pages 275–287, 2015.

[34] Daniel Firestone, Andrew Putnam, Sambrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian M. Caulfield, Eric S. Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt

Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert G. Greenberg. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *Proceedings of the 15th Symposium on Networked Systems Design and Implementation (NSDI)*, pages 51–66, 2018.

[35] Ronald Aylmer Fisher. Statistical methods for research workers. In *Breakthroughs in statistics: Methodology and distribution*, pages 66–70. Springer, 1970.

[36] Brad Fitzpatrick. Distributed caching with memcached. *Linux Journal*, 2004(124):5, aug 2004.

[37] Peter X. Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. Network requirements for resource disaggregation. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*, pages 249–264, 2016.

[38] Peter Xiang Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. pHost: distributed near-optimal datacenter transport over commodity network fabric. In *Proceedings of the 2015 ACM Conference on Emerging Networking Experiments and Technology (CoNEXT)*, pages 1:1–1:12, 2015.

[39] Vasilis Gavrielatos, Antonios Katsarakis, Vijay Nagarajan, Boris Grot, and Arpit Joshi. Kite: Efficient and Available Release Consistency for the Datacenter. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '20, pages 1–16, New York, NY, USA, 2020. Association for Computing Machinery.

[40] Vasilis Gavrielatos, Nicolai Oswald, Antonios Katsarakis, Boris Grot, Arpit Joshi, and Vijay Nagarajan. Scale-Out ccNUMA: Exploiting Skew with Strongly Consistent Caching. *Proceedings of the 13th EuroSys Conference, EuroSys 2018*, 2018-Janua:21:1—-21:15, 2018.

[41] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. Simon: A simple and scalable method for sensing,

inference and measurement in data center networks. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019*, pages 549–564, 2019.

[42] Ali Ghodsi, Matei Zaharia, Scott Shenker, and Ion Stoica. Choosy: Max-min fair sharing for datacenter jobs with constraints. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 365–378, 2013.

[43] Apache Giraph. Apache giraph. Available at `http://giraph.apache.org/`.

[44] S. Jamaloddin Golestani. Network Delay Analysis of a Class of Fair Queueing Algorithms. *IEEE J. Sel. Areas Commun.*, 13(6):1057–1070, 1995.

[45] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. GraphX: Graph Processing in a Distributed Dataflow Framework. In *Proceedings of the 11th Symposium on Operating System Design and Implementation (OSDI)*, pages 599–613, 2014.

[46] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677*, 2017.

[47] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *CoRR*, abs/1706.02677, 2017.

[48] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 51–62, 2009.

[49] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. Efficient memory disaggregation with Infiniswap. *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017*, pages 649–667, 2017.

[50] Chuanxiong Guo, Guohan Lu, Helen J. Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, and Yongguang Zhang. SecondNet: a data center network virtualization architecture with bandwidth guarantees. In *Proceedings of the 2010 ACM Conference on Emerging Networking Experiments and Technology (CoNEXT)*, page 15, 2010.

[51] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi Wei Lin, and Varugis Kurien. Pingmesh: A Large-scale system for data center network latency measurement and analysis. In *SIGCOMM 2015 - Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, volume 45, pages 139–152. ACM, 2015.

[52] Ellen L. Hahne. Round-Robin Scheduling for Max-Min Fairness in Data Networks. *IEEE J. Sel. Areas Commun.*, 9(7):1024–1039, 1991.

[53] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, Marcin Wójcik, and Marcin Wojcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *SIGCOMM 2017 - Proceedings of the 2017 Conference of the ACM Special Interest Group on Data Communication*, pages 29–42, 2017.

[54] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the ACM SIGCOMM 2017 Conference*, pages 29–42, 2017.

[55] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *ICDE Workshops*, pages 41–51, 2010.

[56] P&S Intelligence. Data center market to surpass $343.6 billion revenue by 2030. Available at `https://www.prnewswire.com/news-releases/data-center-market-to-surpass-343-6-billion-revenue-by-2030--says-ps-intelligence-301529854.html`.

[57] Keon Jang, Justine Sherry, Hitesh Ballani, and Toby Moncaster. Silo: Predictable message latency in the cloud. In Steve Uhlig, Olaf Maennel, Brad Karp, and Jitendra Padhye, editors, *Proceedings of the 2015 ACM Conference on Special*

Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015, pages 435–448. ACM, 2015.

[58] Vimalkumar Jeyakumar, Mohammad Alizadeh, David Mazières, Balaji Prabhakar, Albert G. Greenberg, and Changhoon Kim. EyeQ: Practical Network Performance Isolation at the Edge. In *Proceedings of the 10th Symposium on Networked Systems Design and Implementation (NSDI)*, pages 297–311, 2013.

[59] Saurabh Jha, Archit Patke, Ann Gentile, Benjamin Lim, Mike Showerman, Greg Bauer, Supercomputing Applications, Larry Kaplan, Zbigniew Kalbarczyk, Saurabh Jha, Archit Patke, Jim Brandt, Ann Gentile, Benjamin Lim, Mike Showerman, Greg Bauer, Larry Kaplan, Zbigniew Kalbarczyk, William Kramer, and Ravi Iyer. Measuring Congestion in High-Performance Datacenter Interconnects. *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020.

[60] Steven G. Johnson. *The NLopt nonlinear-optimization package*, 2011.

[61] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Using RDMA efficiently for key-value services. *Computer Communication Review*, 44(4):295–306, 2015.

[62] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Design guidelines for high performance RDMA systems. *Proceedings of the 2016 USENIX Annual Technical Conference, USENIX ATC 2016*, pages 437–450, 2016.

[63] Anuj Kalia, Michael Kaminsky, and David G. Andersen. FaSST: Fast, scalable and simple distributed transactions with two-sided (RDMA) datagram RPCs. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*, pages 185–201, Savannah, GA, 2016. {USENIX} Association.

[64] M. R. Siavash Katebzadeh, Paolo Costa, and Boris Grot. Evaluation of an InfiniBand Switch: Choose Latency or Bandwidth, but Not Both. In *ISPASS20*, pages 180–191, 2020.

[65] M. R. Siavash Katebzadeh, Paolo Costa, and Boris Grot. Smart Priority Assignment in Datacenter Networks. In *the 2nd Young Architect Workshop (YArch)*, 2020.

[66] M.R. Siavash Katebzadeh, Paolo Costa, and Boris Grot. Saba: Rethinking datacenter network allocation from application's perspective. In *Proceedings of the Eighteenth European Conference on Computer Systems*, EuroSys '23, page 623–638, New York, NY, USA, 2023. Association for Computing Machinery.

[67] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina, S. Lopez-Buedo, Q. Chen, M. Nemirovsky, D. Roca, H. Klos, and T. Berends. Rack-scale disaggregated cloud data centers: The dReDBox project vision. In *Proceedings of the 2016 Design, Automation and Test in Europe Conference and Exhibition, DATE 2016*, DATE '16, pages 690–695, San Jose, CA, USA, 2016. EDA Consortium.

[68] Antonis Katsarakis, Vasileios Gavrielatos, M.R. Siavash Katebzadeh, Arpit Joshi, Aleksandar Dragojevic, Boris Grot, and Vijay Nagarajan. Hermes: a fast, fault-tolerant and linearizable replication protocol. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 12 2019.

[69] Marios Kogias, Stephen Mallon, and Edouard Bugnion. Lancet: A self-correcting latency measuring tool. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 881–896, Renton, WA, July 2019. USENIX Association.

[70] Marcel Kornacker, Alexander Behm, Victor Bittorf, Taras Bobrovytsky, Casey Ching, Alan Choi, Justin Erickson, Martin Grund, Daniel Hecht, Matthew Jacobs, Ishaan Joshi, Lenni Kuff, Dileep Kumar, Alex Leblang, Nong Li, Ippokratis Pandis, Henry Robinson, David Rorke, Silvius Rus, John Russell, Dimitris Tsirogiannis, Skye Wanderman-Milne, and Michael Yoder. Impala: A Modern, Open-Source SQL Engine for Hadoop. In *Proceedings of the 7th Biennial Conference on Innovative Data Systems Research (CIDR)*, 2015.

[71] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Mike Ryan, David J. Wetherall, and Amin Vahdat. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *Proceedings of the ACM SIGCOMM 2020 Conferenced*, 2020.

[72] Katrina LaCurts, Jeffrey C. Mogul, Hari Balakrishnan, and Yoshio Turner. Cicada:

Introducing Predictive Guarantees for Cloud Networks. In *Proceedings of the 6th workshop on Hot topics in Cloud Computing (HotCloud)*, 2014.

[73] Fan Lai, Jie You, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Sol: Fast Distributed Computation Over Slow Networks. In *Proceedings of the 17th Symposium on Networked Systems Design and Implementation (NSDI)*, pages 273–288, 2020.

[74] Vinh The Lam, Sivasankar Radhakrishnan, Rong Pan, Amin Vahdat, and George Varghese. Netshare and stochastic netshare: predictable bandwidth allocation for data centers. *Comput. Commun. Rev.*, 42(3):5–11, 2012.

[75] Jeng Farn Lee, Meng Chang Chen, and Yeali S. Sun. WF. *Comput. Networks*, 51(6):1403–1420, 2007.

[76] Jeongkeun Lee, Myungjin Lee, Lucian Popa, Yoshio Turner, Sujata Banerjee, Puneet Sharma, and Bryan Stephenson. {CloudMirror}:{Application-Aware} bandwidth reservations in the cloud. In *5th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 13)*, 2013.

[77] Tsern-Huei Lee. Correlated token bucket shapers for multiple traffic classes. In *IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004*, volume 7, pages 4672–4676 Vol. 7, 2004.

[78] Andrew Lester. Application-Aware Bandwidth Scheduling for Data Center Networks. 7(3):194–205, 2014.

[79] Colin Lewis-Beck and Michael Lewis-Beck. *Applied regression: An introduction*, volume 22. Sage publications, 2015.

[80] Shigang Li, Tal Ben-Nun, Salvatore Di Girolamo, Dan Alistarh, and Torsten Hoefler. Taming Unbalanced Training Workloads in Deep Learning with Partial Collective Operations. *arXiv preprint arXiv:1908.04207*, 2019.

[81] Shigang Li, Tal Ben-Nun, Salvatore Di Girolamo, Dan Alistarh, and Torsten Hoefler. Taming unbalanced training workloads in deep learning with partial collective operations. In *PPoPP20*, pages 45–61, 2020.

[82] Hyeontaek Lim, Bin Fan, David G. Andersen, and Michael Kaminsky. SILT: A memory-efficient, high-performance key-value store. In *SOSP'11 - Proceedings*

of the 23rd ACM Symposium on Operating Systems Principles, SOSP '11, pages 1–13, New York, NY, USA, 2011. ACM.

[83] Hyeontaek Lim, Dongsu Han, David G. Andersen, and Michael Kaminsky. MICA: A holistic approach to fast in-memory key-value storage. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014*, pages 429–444, Seattle, WA, 2014. {USENIX} Association.

[84] Yucheng Low, Joseph E. Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. GraphLab: A New Framework For Parallel Machine Learning. *CoRR*, abs/1408.2041, 2014.

[85] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[86] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD Conference*, pages 135–146, 2010.

[87] Yandong Mao, Eddie Kohler, and Robert Morris. Cache craftiness for fast multicore key-value storage. In *EuroSys'12 - Proceedings of the EuroSys 2012 Conference*, EuroSys '12, pages 183–196, New York, NY, USA, 2012. ACM.

[88] Mellanox. ConnectX®-4 VPI IC PRODUCT BRIEF. [Online]. Available: http://www.mellanox.com/related-docs/prod_silicon/PB_ConnectX-4_VPI_IC.pdf.

[89] Mellanox. IB flit simulator.

[90] Mellanox. Mellanox SwitchX and SwitchX®-2 1U Switch and Gateway Systems Hardware User Manual.

[91] Mellanox. SparkRDMA.

[92] Christopher Mitchell, Kate Montgomery, Lamont Nelson, Siddhartha Sen, and Jinyang Li. Balancing CPU and network in the cell distributed b-tree store. In *Proceedings of the 2016 USENIX Annual Technical Conference, USENIX ATC 2016*, pages 451–464, Denver, CO, 2016. {USENIX} Association.

[93] Radhika Mittal. Revisiting Network Support for RDMA Rise of RDMA in datacenters. 1(ii):1–10.

[94] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John K. Ousterhout. Homa: a receiver-driven low-latency transport protocol using network priorities. In *Proceedings of the ACM SIGCOMM 2018 Conference*, pages 221–235, 2018.

[95] Daniel Müllner. fastcluster: Fast hierarchical, agglomerative clustering routines for r and python. *Journal of Statistical Software*, 53(1):1–18, 2013.

[96] Aisha Mushtaq, Radhika Mittal, James Mccauley, Mohammad Alizadeh, Sylvia Ratnasamy, Scott Shenker, U C Berkeley, Radhika Mittal, Sylvia Ratnasamy, James Mccauley, and Scott Shenker. Datacenter congestion control: Identifying what is essential and making it practical. *Computer Communication Review*, 49(3):32–38, 2019.

[97] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, 2019.

[98] OFED. OFED.

[99] OFED. perftest. [Online]. Available: https://github.com/linux-rdma/perftest.

[100] OFED. qperf. [Online]. Available: https://github.com/linux-rdma/qperf.

[101] Gabriele Paoloni. How to Benchmark Code Execution Times on Intel ® IA-32 and IA-64 Instruction Set Architectures. *Intel Manual*, 123(September):1–37, 2010.

[102] Antonis Papadimitriou, Ranjita Bhagwan, Nishanth Chandran, Ramachandran Ramjee, Andreas Haeberlen, Harmeet Singh, Abhishek Modi, and Saikrishna Badrinarayanan. Big Data Analytics over Encrypted Datasets with Seabed. In *Proceedings of the 12th Symposium on Operating System Design and Implementation (OSDI)*, pages 587–602, 2016.

[103] Abhay K Parekh and Robert G Gallager. A generalized processor sharing approach to flow control in integrated services networks: The multiple node case. *IEEE/ACM transactions on networking*, 2(2):137–150, 1994.

[104] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass. *ACM SIGCOMM Computer Communication Review*, 44(4):307–318, 2014.

[105] Lucian Popa, Gautam Kumar, Mosharaf Chowdhury, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica. FairCloud: sharing the network in cloud computing. In *Proceedings of the ACM SIGCOMM 2012 Conference*, pages 187–198, 2012.

[106] Lucian Popa, Praveen Yalagandula, Sujata Banerjee, Jeffrey C. Mogul, Yoshio Turner, and Jose Renato Santos. ElasticSwitch: practical work-conserving bandwidth guarantees for cloud computing. In *Proceedings of the ACM SIGCOMM 2013 Conference*, pages 351–362, 2013.

[107] Diana Andreea Popescu. Latency-driven performance in data centres. Technical Report UCAM-CL-TR-937, University of Cambridge, Computer Laboratory, June 2019.

[108] Diana Andreea Popescu and Andrew W. Moore. PTPmesh: Data Center Network Latency Measurements Using PTP. In *Proceedings - 25th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS 2017*, pages 73–79. IEEE, sep 2017.

[109] Diana Andreea Popescu and Andrew W. Moore. No delay: Latency-driven, application performance-aware, cluster scheduling, 2019.

[110] Diana Andreea Popescu, Noa Zilberman, Andrew W Moore, Diana Andreea, Popescu Noa, Zilberman Andrew, and W Moore. Characterizing the impact of network latency on cloud-based applications' performance. (914):3–20, 2017.

[111] Henrique Rodrigues, Jose Renato Santos, Yoshio Turner, Paolo Soares, and Dorgival Guedes. Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. In *3rd Workshop on I/O Virtualization (WIOV 11)*, 2011.

[112] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the Social Network's (Datacenter) Network. In *Proceedings of the ACM SIGCOMM 2015 Conference*, pages 123–137, 2015.

[113] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan R. K. Ports, and Peter Richtárik. Scaling Distributed Machine Learning with In-Network Aggregation. *arXiv preprint arXiv:1903.06701*, 2019.

[114] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan R. K. Ports, and Peter Richtárik. Scaling Distributed Machine Learning with In-Network Aggregation. In *NSDI21*, pages 785–808, 2021.

[115] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In *IEEE Symposium on Security and Privacy*, pages 38–54, 2015.

[116] WP Dev Shed. How many people use google in 2022? Available at `https://wpdevshed.com/how-many-people-use-google/`.

[117] Social Shepherd. 20 essential meta statistics you need to know in 2022. Available at `https://thesocialshepherd.com/blog/meta-statistics`.

[118] Alan Shieh, Srikanth Kandula, Albert G. Greenberg, and Changhoon Kim. Seawall: Performance Isolation for Cloud Datacenter Networks. In *Proceedings of the 2nd workshop on Hot topics in Cloud Computing (HotCloud)*, 2010.

[119] Madhavapeddi Shreedhar and George Varghese. Efficient fair queueing using deficit round robin. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 231–242, 1995.

[120] Vishal Shrivastav, Ki Suh Lee, Asaf Valadarsky, Han Wang, Hitesh Ballani, Rachit Agarwal, Paolo Costa, Hakim Weatherspoon, Ki Suh Lee, Waltz Networks, Han Wang, Barefoot Networks, Rachit Agarwal, Hakim Weatherspoon, Implementation Nsdi, Ki Suh Lee, Han Wang, Paolo Costa, and Hakim Weatherspoon. Shoal: A Network Architecture for Disaggregated Racks. *Proceedings of*

the 16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, pages 255–270, 2019.

[121] Liang Shuang, Ranjit Noronha, and Dhabaleswar K. Panda. Swapping to remote memory over InfiniBand: An approach using a high performance network block device. *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, page nil, 2005.

[122] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *MSST10*, pages 1–10, 2010.

[123] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST2010*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[124] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of clos topologies and centralized control in acenter network. In *SIGCOMM 2015 - Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 183–197, 2015.

[125] Anirudh Sivaraman, Suvinay Subramanian, Anurag Agrawal, Sharad Chole, Shang Tse Chuang, Tom Edsall, Mohammad Alizadeh, Sachin Katti, Nick McKeown, and Hari Balakrishnan. Towards programmable packet scheduling. *Proceedings of the 14th ACM Workshop on Hot Topics in Networks, HotNets-XIV 2015*, 2015.

[126] Chakchai So-In, Raj Jain, and Jinjing Jiang. Enhanced forward explicit congestion notification (e-fecn) scheme for datacenter ethernet networks. In *2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, pages 542–546, 2008.

[127] Tim Szigeti, Christina Hattingh, Robert Barton, and Kenneth Briley Jr. *End-to-End QoS network design: Quality of Service for rich-media & cloud networks*. Cisco press, 2013.

[128] TechJury. 47 amazon statistics. Available at https://techjury.net/blog/amazon-statistics/.

[129] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156, 2014.

[130] D. Ustiugov, Plamen Petrov, M. R. Siavash Katebzadeh, and Boris Grot. Bankrupt Covert Channel: Turning Network Predictability into Vulnerability. In *Proceedings of the 14th USENIX Workshop on Offensive Technologies (WOOT), co-located with USENIX Security*, 2020.

[131] Balajee Vamanan, Jahangir Hasan, and T. N. Vijaykumar. Deadline-aware datacenter tcp (D2TCP). In *Proceedings of the ACM SIGCOMM 2012 Conference*, pages 115–126, 2012.

[132] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. Ernest: Efficient performance prediction for Large-Scale advanced analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 363–378, Santa Clara, CA, March 2016. USENIX Association.

[133] Jerome Vienne, Jitong Chen, Md Wasi-ur Rahman, Nusrat S. Islam, Hari Subramoni, and Dhabaleswar K. Panda. Performance analysis and evaluation of InfiniBand FDR and 40GigE RoCE on HPC and cloud computing systems. In *Proceedings - 2012 IEEE 20th Annual Symposium on High-Performance Interconnects, HOTI 2012*, pages 48–55, 2012.

[134] Di Xie, Ning Ding, Y Charlie Hu, and Ramana Kompella. The only constant is change: Incorporating time-varying network reservations in data centers. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 199–210, 2012.

[135] Di Xie, Ning Ding, Y. Charlie Hu, and Ramana Rao Kompella. The only constant is change: incorporating time-varying network reservations in data centers. In *Proceedings of the ACM SIGCOMM 2012 Conference*, pages 199–210, 2012.

[136] Reynold S. Xin, Josh Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica. Shark: SQL and rich analytics at scale. In *SIGMOD Conference*, pages 13–24, 2013.

[137] Hailong Yang, Alex D. Breslow, Jason Mars, and Lingjia Tang. Bubble-flux: precise online QoS management for increased utilization in warehouse scale computers. In *Proceedings of the 40th International Symposium on Computer Architecture (ISCA)*, pages 607–618, 2013.

[138] Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. Image Classification at Supercomputer Scale. *CoRR*, abs/1811.06992, 2018.

[139] Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. Image Classification at Supercomputer Scale. In *NeurIPS*, 2018.

[140] Yifei Yuan, Anduo Wang, Rajeev Alur, and Boon Thau Loo. On the feasibility of automation for bandwidth allocation problems in data centers. In *2013 Formal Methods in Computer-Aided Design*, pages 42–45, 2013.

[141] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache Spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65, 2016.

[142] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, oct 2016.

[143] Lixia Zhang. Virtual clock: A new traffic control algorithm for packet switching networks. In *Proceedings of the ACM symposium on Communications architectures & protocols*, pages 19–29, 1990.

[144] Yiwen Zhang, Juncheng Gu, Youngmoon Lee, Mosharaf Chowdhury, and Kang G. Shin. Performance isolation anomalies in RDMA. In *KBNets 2017 - Proceedings of the 2017 Workshop on Kernel-Bypass Networks, Part of SIGCOMM 2017*, KBNets '17, pages 43–48, New York, NY, USA, 2017. ACM.

[145] Yunqi Zhang, David Meisner, Jason Mars, and Lingjia Tang. Treadmill: Attributing the Source of Tail Latency through Precise Load Testing and Statistical Inference. *Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016*, pages 456–468, 2016.

[146] Haishan Zhu, David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Mattan Erez. Kelp: QoS for Accelerated Machine Learning Systems. In *HPCA19*, pages 172–184, 2019.

[147] Haishan Zhu, David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Mattan Erez. Kelp: QoS for accelerated machine learning systems. In *Proceedings - 25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019*, pages 172–184. IEEE, 2019.

[148] Jing Zhu, Dan Li, Jianping Wu, Hongnan Liu, Ying Zhang, and Jingcheng Zhang. Towards bandwidth guarantee in multi-tenancy cloud computing networks. In *2012 20th IEEE International Conference on Network Protocols (ICNP)*, pages 1–10. IEEE, 2012.

[149] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, Ming Zhang, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale RDMA deployments. *SIGCOMM 2015 - Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 45(5):523–536, aug 2015.

[150] Noa Zilberman, Yury Audzevich, G. Adam Covington, and Andrew W. Moore. NetFPGA SUME: Toward 100 Gbps as Research Commodity. *IEEE Micro*, 34(5):32–41, 2014.

[151] Noa Zilberman, Matthew Grosvenor, Diana Andreea Popescu, Neelakandan Manihatty-Bojan, Gianni Antichi, Marcin Wójcik, and Andrew W. Moore. Where has my time gone? In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10176 LNCS, pages 201–214, 2017.

[152] Vojislav Ðukić, Sangeetha Abdu Jyothi, Bojan Karlaš, Muhsen Owaida, Ce Zhang, and Ankit Singla. Is advance knowledge of flow sizes a plausi-

ble assumption? *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019*, pages 565–580, 2019.