# Optimizing Data Analytics Jobs with Fairness Constraints in Datacenter Networks

by

Li Chen

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
The Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto

# Abstract

Optimizing Data Analytics Jobs with Fairness Constraints in Datacenter Networks

Li Chen
Doctor of Philosophy
The Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto
2018

Thanks to the exponential growth of data that needs to be processed in cloud datacenters, data parallel frameworks, such as MapReduce and Spark, have emerged as foundations of cloud computing. It becomes increasingly significant to improve the performance of data analytics jobs running in a shared cluster. Towards this objective, we investigate important research problems in this dissertation as follows.

**Utility-Optimal Coflow Scheduling.** A *coflow* represents a set of network flows in the communication stage of a data parallel job. The completion time of a job is determined by the collective behavior of a coflow and influenced by the amount of network bandwidth allocated to it. We focus on the design and implementation of a new utility optimal scheduler across competing coflows, to provide differential treatment to coflows with different degrees of sensitivity, yet still satisfying max-min fairness across these coflows.

**Job Scheduling across Datacenters.** As large volumes of data are generated and stored across geographically distributed datacenters, recent research proposed to distribute the tasks of a data analytics job across datacenters, considering both data locality and network bandwidth. Yet, it remains an open problem in the more general case, where multiple jobs need to fairly share the resources at these datacenters. We study the assignment of tasks belonging to multiple jobs across datacenters, with the specific objective of achieving *max-min fairness* across jobs sharing these datacenters, in terms of their job completion times.

**Multi-Path Routing across Datacenters.** To process geo-distributed data, optimizing the network transfer in communication stages also becomes increasingly crucial to application performance, as the inter-datacenter links have much lower bandwidth than intra-datacenter links. We focus on exploiting the flexibility of multi-path routing for inter-datacenter flows of data analytics jobs, with the hope of better utilizing inter-datacenter links and thus improve job performance.

We formulate the scheduling problems as lexicographical optimization problems and address the challenges incurred by their multi-objective and discrete nature. We design and implement our solutions as real-world coflow scheduler based on the Varys open-source framework and fair scheduler based on Spark. We also design and implement an optimal multi-path routing and scheduling strategy.

To my family

# Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisor, Professor Baochun Li, for his invaluable guidance and continued support throughout my Ph.D. research towards this thesis. His passion for research and his insights to research topics are inspiring. The sense of independent thinking and the skills of presentation and writing I learned from him are lifelong beneficial.

I would also like to thank Professor Bo Li, who enlightened me with his research vision and provided valuable suggestions throughout the work. I am grateful for my Ph.D. supervisory committee members, Professor Ben Liang and Arno Jacobsen, and all other oral examination committee members, Professor Ding Yuan, Lacra Pavel and Qun Li, for their valuable feedback and detailed suggestions on revising the thesis to its final completion.

It is an enjoyable experience working with all the members in the iQua Research Group at University of Toronto: Chen Feng, Yuan Feng, Zimu Liu, Hong Xu, Wei Wang, Jun Li, Shuhao Liu, Liyao Xiang, Weiwei Fang, Xu Yuan, Zhiming Hu, Yanjiao Chen, Hao Wang, Zhiping Cai, Xiaoyan Yin, Hongyu Huang, Wanyu Lin, Yinan Liu, Wenxin Li, Siqi Ji, Yilun Wu, Jingjie Jiang, Shiyao Ma, Jiapin Lin, Tracy Cheng, Jiayue Li, Yuanxiang Gao, Chen Ying, Yifan Gong. I am grateful for their wisdom that inspired me, for their humor that made life joyful, and for their kindness to offer help in various aspects. Especially, I would like to thank Shuhao Liu and Liyao Xiang, who are always ready to offer help and suggestions, both in research and in daily life. I would also like to express my gratitude to my roommate and good friend Weiwei Li. Many thanks to each of you for our friendship along my Ph.D. journey and I always remember the good memories we have enjoyed in different places.

Finally, I would like to express my love and gratitude to my family for their unconditional love and continuous encouragement. I always feel touched and inspired by my parents being brave, strong and optimistic when faced with adversities in life. I always feel grateful for the sincere concern and support from my relatives that accompany me through difficulties. Accomplishing this work was not possible without the unconditional support from my beloved family members.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Thanks to the exponential growth of data that needs to be processed in cloud datacenters, data parallel frameworks, such as MapReduce [1] and Spark [2], have emerged as foundations of cloud computing. In data parallel frameworks, a *job* typically proceeds in consecutive computation *stages*; and each of these stages consists of a number of computation *tasks* that are processed in parallel. Before the next computation stage may begin, multiple network flows need to be initiated in parallel to transfer intermediate data from the preceding stage. Due to the volume of intermediate data to be transferred, such network transfers across stages may have a significant impact on the performance of typical jobs in big data processing. It has been shown that they usually account for more than 50% of entire job completion times [3].

To optimize performance of data analytics jobs in data parallel frameworks, we focus on the optimal allocation of network and computation resources among concurrent jobs in a shared cluster. We start with the allocation of link bandwidth among flows from multiple jobs sharing the datacenter network. The guideline towards performance-centric fairness, where the performance is inversely related to the job completion time, is proposed and studied, followed by the fairness with respect to utilities that better represent distinctive job requirements. We then extend our consideration to a global cluster consisting of geo-distributed datacenters. The allocation of computing slots in each datacenter and the diversity of path selection among datacenters are respectively incorporated in our resource allocation among concurrent jobs. Detailed introduction of each problem is presented as follows.

In the context of a privately operated datacenter shared by network-intensive applications or jobs, how should the critical *link bandwidth* be shared among these jobs? It is commonly accepted in the literature that bandwidth should be shared in a *fair* manner (*e.g.*, [4]), yet there has been no general consensus on how the notion of *fairness* should be defined. The traditional wisdom on fair bandwidth sharing has largely focused on datacenters in a public cloud, where virtual machines (VMs) are used to host applications for the tenants. For example, bandwidth on a link can be allocated fairly across different flows, VM pairs, or tenants (according to their payments). In this dissertation, we argue that the notions of fairness proposed in the literature are not applicable to the context of data parallel jobs sharing a private datacenter. Rather than being fair across competing flows or tenants according to their payments, we propose the notion of *performance-centric fairness*, in that fairness should be maintained with respect to the *performance* across multiple data parallel jobs.

But how, after all, shall we rigorously define the notion of *performance-centric fairness*? As available bandwidth resources are allocated for data parallel jobs to transfer data in their communication stages, their performance is best represented by the amount of time needed to complete the data transfer, called the *transfer time.* To achieve their best possible performance with the shortest possible transfer times, the guiding principle of *weighted performance-centric fairness* is that the *reciprocal* of the transfer times should be proportional to their weights across competing jobs. To put it simply, jobs with equal weights sharing the same private datacenter should enjoy the same performance. In this dissertation, our first work is to **study the bandwidth allocation problem to optimize job performance constrained by performance-centric fairness, and explore the tradeoff between fairness and efficiency.**.

Since a network transfer of a data analytics job is not considered complete till all of its constituent flows have finished, it is the collective behavior of all of these flows that matters, rather the individual behavior of each flow. These flows are hence referred to as a *coflow* [5]. In a MapReduce job, for example, a coflow consists of all the flows in the *shuffle* phase, which transfers intermediate data from *mapper* tasks to *reducer* tasks. As the datacenter network is shared by active coflows from multiple competing jobs, it is critical to schedule these coflows efficiently and fairly. Network resources should be allocated at the level of *coflows* — rather than individual flows — in order to achieve the best possible performance with respect to job completion times.

However, due to their inherent nature, different jobs have widely diverging requirements with respect to their completion times: an interactive query in a web application should not be similarly treated as a background job for data analytics. Intuitively, we may use different *utility functions* [6] to model such a diverging range of sensitivity to job completion times. Existing research efforts on coflow scheduling focused on minimizing coflow completion times [7, 8, 9] and meeting coflow deadlines [7]. All coflows were treated identically as equal citizens in a datacenter, and the average coflow completion time was to be minimized. In this dissertation, we depart from such conventional wisdom, and argue that more time-sensitive coflows should be allocated more network resources, allowing them to complete earlier, achieving a higher utility based on their utility functions. Of course, such differential treatment may negatively affect the performance of background jobs that tolerate longer completion times. Therefore, from a global perspective and in the general case, we will need to **allocate resources optimally across all concurrent coflows, such that they will achieve their best possible utilities, and max-min fairness across coflows can be achieved**.

Beyond the scope of a single datacenter, it is increasingly common for large volumes of data to be generated, stored and processed in a geographically distributed fashion, across multiple datacenters around the world. For example, global services deployed by Microsoft and Google [10, 11] routinely span multiple geo-distributed datacenters and generate large volumes of data for analysis, such as user activity and system monitoring logs.

Traditionally, a data parallel job runs within a single datacenter, where all its input data are stored. Network flows generated during data processing are bounded within the intra-datacenter network. However, when input data are distributed across multiple datacenters, it is inevitable that flows across datacenters will be generated to traverse inter-datacenter links. To process such geo-distributed data, a naive approach is to gather all the data to be processed locally within a single datacenter. Naturally, transferring huge amounts of data across datacenters may be slow and inefficient, since bandwidth on inter-datacenter network links is limited [12].

Existing research (*e.g.*, [13, 14]) has shown that better performance can be achieved if tasks in an analytics job can be distributed across datacenters, and located closer to the data to be processed. In this case, designing the best possible task assignment strategy to assign tasks to datacenters is important, since different strategies lead to different flow patterns across datacenters, and ultimately, different job completion times.

When designing optimal task assignment strategies, however, existing works in the literature [13, 14] only considered a single data analytics job. The problem of assigning tasks belonging to multiple jobs across datacenters remains open. Given the limited amount of resources at each datacenter, multiple jobs are inherently competing for resources with each other. It is, therefore, important to maintain *fairness* when allocating such a shared pool of resources, which cannot be achieved if tasks from one job are assigned without considering the other jobs. As such, we propose a **new task assignment strategy that is designed to achieve max-min fairness across multiple jobs with respect to their performance**, as they compete for the limited pool of shared resources across multiple geo-distributed datacenters. To be more specific, we wish to minimize the job completion times across all concurrent jobs, while maintaining max-min fairness.

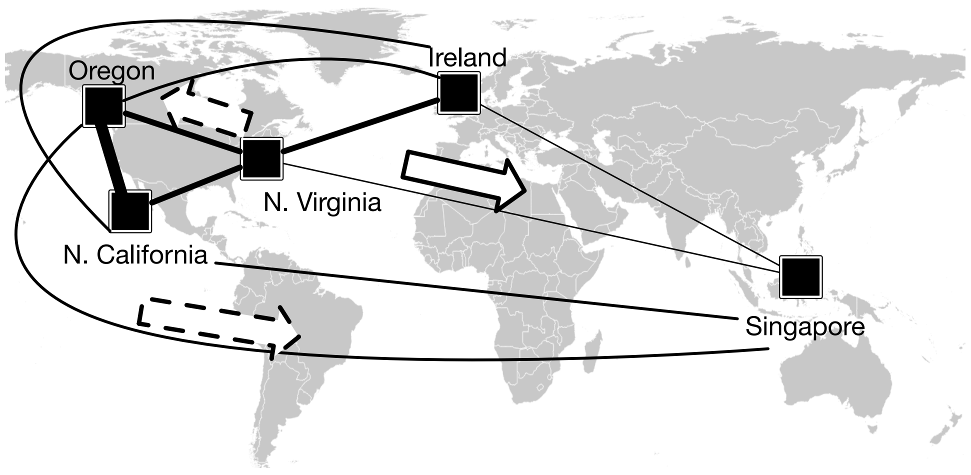


Figure 1.1: Flexibility in routing (direct and detoured paths) for inter-datacenter traffic.

Apart from designing optimal task placement strategies, existing efforts on improving the performance of geo-distributed data analytics jobs also focus on reducing the total amount of cross-datacenter traffic. However, this does not necessarily reduce the job completion times, since the network transfer time is not only influenced by the traffic size, but also the bandwidth of inter-datacenter links along which the flows traverse.

All these existing works seek to alleviate the performance degradation of inter-datacenter transfers by modifying the generated traffic pattern across datacenters, which do not directly solve the problem from the perspective of optimizing the network transfers given certain traffic patterns. To fill this gap, we further propose to **optimize the inter-datacenter transfers by exploiting the path flexibility to better utilize the inter-datacenter link bandwidth**, which provides a complementary and orthogonal way to improve the performance of geo-distributed data analytics jobs.

To have a better intuition of the routing flexibility, we present five datacenters of Amazon EC2 in Fig. 1.1, which are geographically distributed across the world. The thickness of the line between each pair of datacenters is proportional to the amount of available bandwidth. It is clearly shown that the bandwidth of the cross-ocean link between N. Virginia and Singapore is much smaller than any other link. For flows between N. Virginia and Singapore, apart from using the direct path through the narrow link, as illustrated by the solid arrow, we can take the detoured path by first forwarding the traffic to Oregon, which then relays the traffic to Singapore, as illustrated by the dashed arrows in Fig. 1.1. Both links along the detoured path have larger amounts of bandwidth than the direct link. Other detoured paths, such as the one through N. California, can also be considered to accelerate the transfer.

By leveraging such flexibility of multi-path routing, we attempt to make the best utilization of the inter-datacenter links to accelerate geo-distributed data analytics jobs. Different from existing efforts [15, 9] on multi-path routing, which require that a flow can only take one selected path, we allow each flow to be split and routed through multiple paths, so that the bandwidth can be better utilized to improve the job performance. In particular, given a set of inter-datacenter flows generated by data analytics jobs, we coordinate them with respect to their routing paths and sending rates, so as to accelerate the network transfers and eventually achieve the best possible performance for all the jobs.

## 1.1 Contributions

We first study the bandwidth allocation problem guided by our proposed notion of performance-centric fairness, and propose two algorithms to arbitrate the tradeoff between fairness and efficiency, interpreted from two perspectives.

We then formally formulate the problem of maximizing the utilities achieved by each of the coflows as a *lexicographical maximization* problem, which has unique challenges that make it difficult to solve. Scheduling among coflows over a discretized time domain is essentially an integer optimization problem, which in general is NP-hard [16]. In addition, to accommodate a diverse set of practical utility functions (such as a sigmoid function), we do not assume that these utility functions are convex or concave; standard convex optimization techniques are therefore not directly applicable.

To address these challenges, we first consider the subproblem of maximizing the worst utility among all the concurrent coflows, which turns out to have a *totally unimodular* coefficient matrix for linear constraints, based on an in-depth investigation of the problem structure. Such a nice property guarantees that the extreme points in a feasible solution polyhedron are integers. Moreover, with several steps of non-trivial transformations, we show that the optimal solution to the original problem can be obtained by solving an equivalent problem with a *separable convex objective.* With these structures identified, we can then apply the $\lambda$-technique and linear relaxation to obtain a linear programming (LP) problem, which is guaranteed to have the same solution to the original problem. As a result, any LP solver, (*e.g.,* Mosek [17]), can be used for maximizing the utility in each coflow, and to efficiently compute the overall scheduling decisions that achieve the optimal coflow utilities with max-min fairness.

We proceed to design and implement a real-world coflow scheduler that enforces our utility optimal scheduling policy in the Varys [7] coflow scheduling framework. Experimental results demonstrate the effectiveness of our scheduler in optimizing coflow utilities. Compared with the state-of-the-art coflow scheduler which is utility agnostic, our scheduler achieves significant utility improvement, and thus better satisfy job requirements.

We also formally formulate the problem of scheduling jobs across geo-distributed datacenters, with the objective of minimizing job completion times and achieving max-min fairness, as a *lexicographical minimization* problem. Apart from the challenge incurred by the multi-objective nature of lexicographical optimization, the task assignment problem is essentially an integer optimization problem, which in general is NP-hard.

We address these challenges by first considering the subproblem of minimizing the worst (longest) job completion time among all the concurrent jobs. From an in-depth investigation of the problem structure, we again prove that the coefficient matrix for linear constraints is totally unimodular, and the original problem can be equivalently transformed to a problem with a separable convex objective. With these properties, we then apply the $\lambda$-technique and linear relaxation to obtain an LP problem, which is guaranteed to have the same solution as the original problem. To demonstrate the practicality of our proposed solution, we design and implement a new job scheduler to assign tasks from multiple jobs to geo-distributed datacenters, in the context of Apache Spark. Our experimental results on multiple Amazon EC2 datacenters have shown that our new scheduler is effective in optimizing job completion times and achieving max-min fairness.

We further design and implement our multi-path routing and scheduling strategy as a transfer optimization service for geo-distributed data analytics jobs. With a special focus on Spark, we provide a simple and convenient API for it to delegate its traffic. Our transfer service constructs an overlay inter-datacenter network, which follows the principle of Software Defined Networking (SDN) [18] at the application layer. All the inter-datacenter traffic delegated to our service would be fully controlled, following the optimal solution instructed from a centralized controller where our strategy is realized. We evaluate our prototype implementation of such an optimal transfer service with a wide array of real-world experiments, running various Spark jobs over Google Cloud. Compared with the state-of-the-art, our strategy has shown substantially faster shuffle completion time, up to approximately 30%. To the best of our knowledge, we present the first design and implementation of a cloud service for network optimization within geo-distributed data analytics.

## 1.2 Organization

The remainder of this dissertation is organized as follows.

**Chapter 2** reviews related literatures.

**Chapter 3** presents our theoretical study on the problem of bandwidth allocation among sharing jobs, with a tunable degree of relaxation on performance-centric fairness. This chapter is based on our work published in IEEE Transactions on Parallel and

Distributed Systems [19], collaborated with Yuan Feng, Baochun Li and Bo Li.

**Chapter 4** presents our theoretical study on the utility-optimal scheduling problem for competing coflows from data analytics jobs sharing datacenter networks, as well as the design, implementation and evaluation of our utility-optimal scheduler. This chapter is based on our work published in IEEE INFOCOM 2016 [20], collaborated with Wei Cui, Baochun Li and Bo Li.

**Chapter 5** extends our scope to the context of geographically distributed datacenters. With the focus on jointly assigning tasks from multiple data parallel jobs across datacenters, a theoretical study of the job scheduling problem is presented, followed by the design, implementation and evaluation of a fair job scheduler. This chapter is based on our work published in IEEE INFOCOM 2017 [20] and IEEE Transactions on Network Science and Engineering [21], collaborated with Shuhao Liu, Baochun Li and Bo Li.

**Chapter 6** shifts the focus to the direct optimization for network transfer across datacenters, rather than rearranging inter-datacenter traffic patterns. The design of our multi-path routing and scheduling strategy is presented, followed by its implementation and evaluation as an inter-datacenter transfer optimization service. This chapter is based on joint work with Shuhao Liu, Baochun Li and Bo Li.

**Chapter 7** concludes this dissertation, with a summary of our work and a discussion on future directions.

# Chapter 2

# Background and Related Work

For efficient data analytics over a large volume of data, data parallel frameworks, such as MapReduce and Spark, have become the mainstream platform. In such frameworks, the input data of a job is partitioned into multiple splits for parallel processing. Typically, a job proceeds in consecutive computation stages, each of which consists of a number of parallel computation tasks. Between consecutive stages, multiple network flows may need to be initiated in parallel to transfer intermediate data, which is called the communication stage.

It is worth noting that the communication stage does not complete until all of its flows have finished. This indicates that all of the constituent flows share the same application-level performance metric. No matter how fast a single flow finishes, the application-level performance is determined by the slowest flow. The abstraction of *coflow*, representing the collection of such parallel flows in a communication stage, is thus proposed to convey the application-level performance requirements. To be more general, a coflow is defined as a set of flows that share the same performance metric — the coflow completion time, which is the maximum of the flow completion time across all the consisting flows. Each flow within a coflow independently sends a predefined amount of data from a sender to a receiver, which may not be on the same machine.

When there are multiple such data analytics jobs running concurrently in a cluster, they are competing with each other for both the computation and network resources.

Depending on the locations of the input datasets, the cluster for large-scale data analytics jobs can be within a single datacenter or spanning across multiple datacenters. It is important to optimize the resource allocation in the shared cluster, by an optimal design of task assignment and coflow scheduling, to improve the performance of these jobs, while taking fairness into account.

## 2.1 Coflow Scheduling in Datacenter Networks

With data parallel frameworks extensively deployed for big data processing, it has received an increasing amount of research attention to optimize the network allocation among coflows for better job-level performance.

The concept of coflow was first proposed by Chowdhury *et al.*, which defines a collection of flows between successive computation stages accounting for the application semantics [5]. As pointed out, rather than individual flow, coflow should be the basic unit for network optimization. Before the formal proposition of the coflow concept, Orchestra [3] is the preliminary work that is aware of the important characteristic of barrier in optimizing transfers of data parallel applications. It proposed to allocate rates to each flow in a shuffle using weighted fair sharing, where the flow weight is proportional to the volume of data to be sent by this flow.

Varys [7] took the initiative to propose effective heuristics for coflow scheduling, in order to minimize the average coflow completion time (CCT) and meeting coflow deadlines. Baraat [22] and Stream [23] focused on decentralized coflow scheduling. Aalo [24] and CODA [25] studied the scheduling problem without any prior knowledge of coflows. Qiu *et al.* [8] considered the release dates of coflows and proposed the first polynomial-time approximation algorithm to minimize the weighted CCT. Extending coflow awareness into routing, RAPIER [9] and Li *et al.* [15] designed strategies for joint coflow scheduling and routing to minimize the average CCTs.

However, all the existing efforts treat competing coflows equally in minimizing CCTs, or at most differentiate them based on whether they have deadlines or not. Such a lack of fine grained differentiation can not satisfy the diverse requirements of coflows with respect to their CCTs, which can be reflected in their evaluation of completion times as utility values. For example, a user-interactive query job is critical to its completion time, or probably associated with a hard deadline. Failing to complete the job by its target time would result in a small or zero value of utility. In contrast, a background job for data analytics may be completion-time-insensitive, which means that as long as it successfully generates the final result, it would obtain a fixed value of utility, regardless of the time when it completes.

In this dissertation, we capture these requirements with utility functions, which can quantify the degrees of sensitivity to CCTs. Our scheduling objective is to optimize all the coflow utilities with max-min fairness, which means that all the coflows are expected to achieve their best possible utilities without impacting others.

The heterogeneous sensitivity of job completion times were studied in CORA [6], which designed a utility optimal scheduler to allocate computing resource to jobs. Despite sharing a similar philosophy with CORA, our scheduling problem is quite different and more challenging, due to the inherent complexity in network scheduling that involves coupled resources, to be elaborated in Chapter 4.

## 2.2 Geo-Distributed Data Analytics Jobs

As large volumes of data are increasingly generated globally and stored in geographically distributed datacenters, improving performance of data analytics jobs with geo-distributed input data has received an increasing amount of research attention. Based on their objectives, existing efforts can be roughly divided into two categories: reducing the amount of inter-datacenter network traffic to save operation costs, and reducing the job completion time to improve application performance.

Vulimiri *et al.* [26, 12] took the initiative to reduce the amount of data to be moved across datacenters when running geo-distributed data analytics jobs. To reduce the bandwidth cost, they formulated an integer programming problem to optimize the query execution plan and the data replication strategy. They also took advantage of the abundant storage resources to aggressively cache results of queries, to be leveraged by subsequent queries to reduce data transfers. Pixida [27] proposed to divide the DAG of a job into several parts, each to be executed in a datacenter, with the objective of minimizing the total amount of traffic among these divided parts. Gaia [28] proposed a new synchronization model for geo-distributed machine learning to reduce communication overhead across datacenters. Although the inter-datacenter traffic size is reduced by these efforts, it is not guaranteed that the jobs are accelerated, as job completion times also depend on the available bandwidth of inter-datacenter links.

As a representative work in the second category [13, 29, 14], Iridium [13] proposed an online heuristic to place both data and tasks across datacenters. Unfortunately, it assumes that the wide-area network that interconnects datacenters is free of congestion, which is far from realistic. Flutter [14] removed this unrealistic assumption, formulated a lexicographical minimization problem of task assignment for a single stage of one job, and obtained its optimal solution.

However, all existing works focused on assigning tasks in a single job, without considering the inherent competition for resources among concurrent jobs. Accounting for the scenario of multiple jobs sharing geo-distributed datacenters, Hung *et al.* [30] proposed a greedy scheduling heuristic to make job scheduling decisions across geo-distributed datacenters, with an objective of reducing the average job completion time. However, it assumes that the task assignment is predetermined, and the scheduling decision is the execution order of all the assigned tasks in each datacenter. Therefore, despite sharing a similar context of considering multiple jobs sharing the same pool of computing resources in geo-distributed datacenters, this work is orthogonal to our work, which aims to determine the best possible placement for tasks of all the sharing jobs with the consideration of fairness (Chapter 5).

## 2.3 Multi-Path Routing and Scheduling for Coflows

Different from all the existing works above, RAPIER [9] and Li *et al.* [15] proposed to jointly consider coflow routing and scheduling, which is orthogonal to reducing the total size of traffic or modifying the traffic pattern by specifying the task placement. In their solutions, each flow is routed along a single path which is selected from multiple available ones.

In contrast, we allow each flow to be split along multiple paths to better utilize the bandwidth, and calculate the optimal rate assignment, so that the network transfers from all the sharing jobs achieve their best possible completion times (Chapter 6). Moreover, different from their objective of minimizing the average coflow completion time, our objective is to achieve the best possible performance for all the coflows, with fairness considered. The most important merit of our work is that we have implemented our strategy and evaluated with real coflows, rather than simulated [15] or emulated ones [9].

## 2.4 Performance Optimization for Data Analytics Jobs in A Single Datacenter

There are plenty of existing efforts ([31, 32, 33], *etc.*) related to performance optimization in big data analytics frameworks. They proposed task assignment strategies to improve data locality and fairness [31, 32], speculation strategies to mitigate the negative impact of stragglers ([33]), and coflow scheduling strategies ([7, 24, 8], *etc.*) to improve network performance by minimizing the average coflow completion time. However, they are all designed for jobs running in a single datacenter, and do not work effectively in the multi-datacenter scenario. In this dissertation, we design performance optimization strategies for both single-datacenter and multi-datacenter scenarios, with a special focus on optimizing network performance and achieving fairness among competing jobs.

## 2.5 Software-Defined Networking (SDN) in the Wide Area Network

The concept of SDN has been proposed to facilitate the innovation in network control plane [34, 35]. The design principle of SDN is to decouple the control plane, which makes packet forwarding decisions, from the data plane, which consists of hardware switches that forward packets. The centralized programmable controller communicates with the switches through standard SDN protocols such as OpenFlow [36], to form a global view of the network, make centralized decisions and install forwarding rules. In the inter-datacenter wide-area network, SDN has been recently adopted to provide centralized control with elegantly designed traffic engineering strategies [37, 38, 39]. With flexibility and centralized control, better traffic engineering strategies have been designed and implemented across their geo-distributed datacenters. Different from these efforts, our work takes advantage of the SDN principle realized in the application layer, without requiring hardware support. Moreover, our work in this dissertation focuses on improving performance for data analytics jobs with more complex communication patterns, controlling flows at a finer granularity.

## 2.6 Fair Bandwidth Allocation

Bandwidth allocation among multiple tenants in public cloud datacenters has received a substantial amount of recent research attention [40, 41, 4, 42, 43, 44, 45, 46, 47]. The general focus of these works has been on ensuring fair allocation among different tenants according to their payments. For example, NetShare [40] achieves tenant level fairness while Seawall [41] achieves fairness between VM sources. FairCloud [4] allocates bandwidth on congested links based on the weights of the communicating VM-pairs, thus achieving VM-pair level fairness. However, in our setting of a private datacenter running data parallel frameworks, the previous notion of fairness is not applicable.

In the context of a private datacenter, Kumar *et al.* proposed that bandwidth should be allocated with the awareness of the communication patterns of data parallel applications [48]. Their focus is mainly on effective parallelization for each application, *i.e.*, the completion time should be $N$ times faster if the application parallelizes by $N$. However, when tasks of one application share bandwidth with tasks of different applications at different bottlenecks, it is not known what performance each application should expect, without a clear definition of fairness with respect to application performance. In contrast, our proposition of performance-centric fairness [49] fills this gap, and offers a definitive guide to the problem of bandwidth allocation among multiple data parallel applications in a private datacenter.

Such a definition of fairness has been further explored in our following works which jointly considered the flexibility of task placement [50] and path selection [51], respectively. It has been demonstrated that performance-centric fairness has effectively served as a guidance in more complex scenarios in practice, when additional dimensions of problems are involved. Complementary to such a broader extension of the performance-centric fairness, in this dissertation, we choose a deeper extension to investigate the inherent tradeoff between performance-centric fairness and efficiency from two different perspectives. In particular, a new bandwidth allocation algorithm has been designed and evaluated to explore the fairness-efficiency tradeoff thoroughly. With our study, insights can be provided for service providers to deploy their bandwidth allocation strategy at the sweet spot (Chapter 3).

# Chapter 3

# Performance-Centric Fair Bandwidth Allocation

In this chapter, we investigate the allocation of link bandwidth shared among applications or jobs running data parallel frameworks, such as MapReduce, in private datacenters. We first introduce the rigorous definition of *performance-centric fairness*, with the guiding principle that the performance that data parallel applications will enjoy should be proportional to their weights, and study the problem of maximizing application performance while maintaining strict performance-centric fairness. We then present an inherent trade-off between fairness and efficiency, which is interpreted from the perspectives of bandwidth utilization and social welfare, respectively. From the first perspective, we propose an algorithm to improve bandwidth utilization by introducing an extended version of fairness. From the second perspective, we formulate an optimization problem of bandwidth allocation that maximizes the social welfare across all the applications, allowing a tunable degree of relaxation on performance-centric fairness. A distributed algorithm is then presented to solve the problem, based on dual based decomposition. With extensive simulations, we demonstrate the effectiveness of our algorithms in improving efficiency and application performance (by up to $1.4X$), with flexible degree of relaxation on the performance-centric fairness.

## 3.1 A Case for Performance-Centric Fairness

Data parallel applications partition their input data into multiple splits, which are processed in parallel by computation tasks. As the total amount of computation workload remains the same, a scale-up of $n$ would result in a speed-up of $n$ for the computation stage. To be specific, with $n$ times the computation tasks, the completion time of the computation stage reduces to its $1/n$. However, such a scale-up does not apply to the network transfer time, since the total amount of network traffic may increase with additional parallel tasks, depending on the *communication pattern* between computation tasks.

A typical MapReduce application uses the *shuffle* communication pattern between its map and the reduce tasks, while machine learning applications use a *broadcast* communication pattern [52]. We show an example for both communication patterns in Fig. 3.1. In the base cases without any parallelization in the computation stages, the only computation task in $A$ (or $B$) produces 500 MB of intermediate data, which is directly transmitted to the task $A'$ (or $B'$). In the cases where both applications employ two parallel computation tasks in each computation stage, the input data is then partitioned into two equal splits, and the amount of intermediate data generated by each task is half of the base case. Since $A$ is a MapReduce task with the *shuffle* communication pattern, the data produced by each map task, $A1$ and $A2$, is partitioned into two equal sets, each with a size of 125 MB, to be sent to both $A3$ and $A4$. In contrast, since $B$ uses the *broadcast* communication pattern, each task in the first stage, $B1$ and $B2$, broadcasts all of its produced data to both tasks in the second stage, $B3$ and $B4$.

With the knowledge of the effects of communication patterns on the amount of network traffic, we are now ready to discuss the notion of performance-centric fairness in the context of bandwidth allocation, when $A$ and $B$ share the link bandwidth in a private datacenter as shown in Fig. 3.2. Specifically, $A1, A2$ co-locate with $B1, B2$ on physical machine $P1$, sharing the egress link bandwidth of $P1$ with a capacity of 500 MB/s.

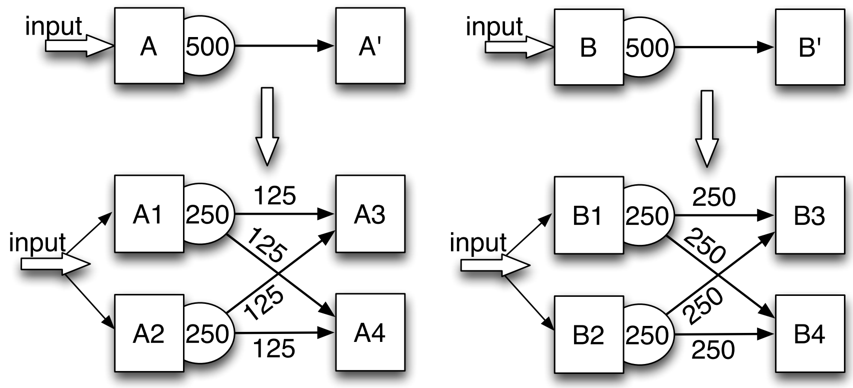


Figure 3.1: The amount of data to be transmitted when parallelizing a MapReduce application with the *shuffle* communication pattern, and a machine learning application with the *broadcast* communication pattern.

Similarly, $A3, A4$ and $B3, B4$ share the ingress link bandwidth (500 MB/s) of $P2$.

For each application, the transfer time is defined as the completion time of the slowest flow among all flows in its communication stage. To be specific, the transfer time is decided by both the amount of network traffic between each task pair and the bandwidth allocated to each flow. According to their importance, $A$ and $B$ are assigned weights of $w_A$ and $w_B$, respectively. To satisfy both applications, *i.e.*, to ensure fairness between $A$ and $B$ with respect to their network performance (or transfer time), the egress bandwidth on $P1$ and the ingress bandwidth on $P2$ should be allocated so that the transfer times represented by $t_A$ and $t_B$ satisfy $\frac{1}{t_A} : \frac{1}{t_B} = w_A : w_B$. In this way, the allocation achieves weighted performance-centric fairness for $A$ and $B$.

To better illustrate this notion, we show two more examples of bandwidth allocation shown in Fig. 3.2. Since the egress link at $P1$ and the ingress link at $P2$ are both shared by $A$ and $B$ in a symmetric way, we use the term *link bandwidth* for both egress and ingress link bandwidth for simplicity. When both $A$ and $B$ have the same weight, the transfer times of $A$ and $B$ should be equal according to weighted performance-centric fairness. Each flow of $A$ is allocated $\frac{125}{3}$ MB/s, thus the transfer time is $125/\frac{125}{3} = 3$s. With $\frac{250}{3}$ MB/s link bandwidth allocated to each flow, $B$ can achieve a transfer time of $250/\frac{250}{3} = 3$s. Since $A$ and $B$ with the same weight enjoy the same performance

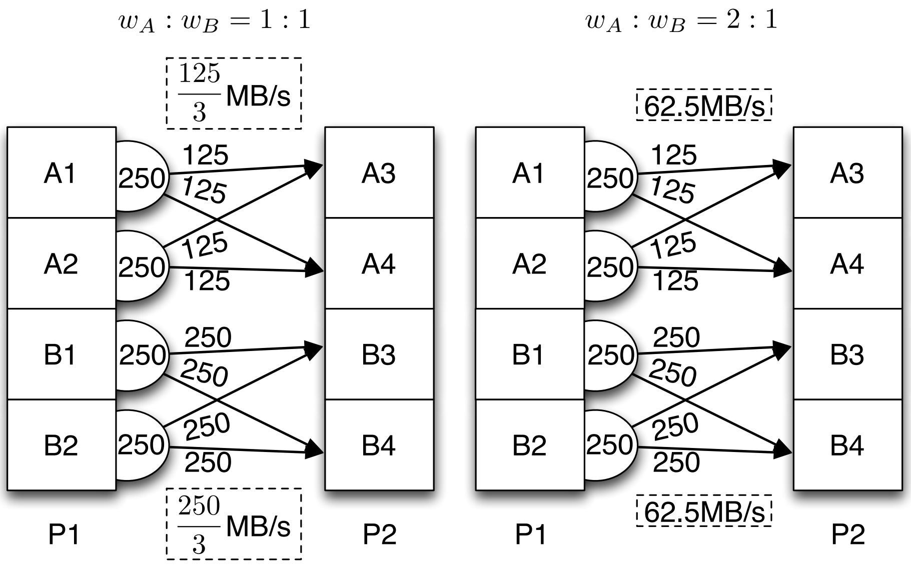


Figure 3.2: Examples of bandwidth allocation achieving weighted performance-centric fairness in two cases: 1) both applications have the same weight; 2) the two applications have different weights.

with respect to their transfer times, this allocation achieves weighted performance-centric fairness between the two applications.

In the case where $A$ and $B$ have different weights, if we allocate 62.5 MB/s to each flow of both applications as shown in Fig. 3.2, the transfer time of $A$ is $\frac{125}{62.5} = 2$s, and the transfer time of $B$ is $\frac{250}{62.5} = 4$s. Since $\frac{1}{2} : \frac{1}{4} = w_A : w_B = 2$, weighted performance-centric fairness is again achieved with this allocation.

We argue that weighted performance-centric fairness best meets the requirements of application performance, in the privately operated datacenters. The major reason is that it directly targets the eventual application performance, rather than the amount of resources allocated to each application as previous notions of fairness. Particularly, if two equally important applications, one with the shuffle pattern and the other with the broadcast pattern, scale up by doubling their parallel computation tasks, they should expect the same degree of performance improvement, regardless of their communication patterns. Such an intuitive performance requirement is achieved with our performance-centric fairness, as demonstrated in Fig. 3.2. In contrast, with traditional fairness, for example, the per-flow fairness, the bandwidth allocated to each flow remains the same, as the application scale-up does not impact the number of sharing flows. However, due to different communication patterns, the amounts data transferred by flows from the two applications become different. Hence, per-flow fairness fails to guarantee the corresponding degree of performance improvement for these applications.

***Defining weighted performance-centric fairness.*** With an intuitive idea of weighted performance-centric fairness in our illustrative examples, we now present a strict definition in a general setting.

In a privately operated datacenter, multiple data parallel applications share the bandwidth resource by co-locating some of their tasks on some of the physical machines. Each application $k \in \mathcal{K} = \{1, 2, ..., K\}$ is assigned the weight $w_k$ according to its importance. If for any application $k$, its performance, defined as the reciprocal of its transfer time $t_k$ achieved under a certain allocation, satisfies the following condition:

$$(1/t_{k1}) : (1/t_{k2}) = w_{k1} : w_{k2}, \quad \forall k1, k2 \in \mathcal{K} \tag{3.1}$$

then weighted performance-centric fairness has been achieved.

Weighted performance-centric fairness is defined with respect to the performance achieved by all applications, rather than the amount of bandwidth resource obtained by each flow or each task. In this sense, this fairness is defined at the level of applications, which is quite different from fairness definitions at the flow level (TCP), VM source level (*e.g.*, [41]), VM-pair level (*e.g.*, [4]) or the tenant level (*e.g.*, [40]) proposed in the literature in the context of datacenters in a public cloud. To achieve such a fairness, the allocation should be aware of the applications' communication patterns, which will affect the amount of network traffic in each flow, and further impact the transfer times of applications.

## 3.2 Allocating Bandwidth to achieve Performance-Centric Fairness

Given the intuitive examples and the strict definition of weighted performance-centric fairness in the previous section, we now study the bandwidth allocation problem with the fairness requirement in a general scenario.

We consider a private datacenter where there are $K$ data parallel applications running concurrently, with their tasks distributed across $N$ physical machines. These applications typically partition the computation among multiple tasks, and communicate the intermediate data between the tasks belonging to different computation stages. The communication pattern can be either *shuffle* as in MapReduce [1], or *broadcast* as in machine learning applications [52].

On each physical machine (or server interchangeably) $n \in \mathcal{N} = \{1, 2, ..., N\}$, tasks from different applications will share its link bandwidth, including both the egress link with capacity $B_n^E$ and the ingress link with capacity $B_n^I$. Since the bisection bandwidth in datacenter networks has been significantly improved by multi-path routing (*i.e.*, [53]) and multi-tree topologies (*i.e.*, [54]), we assume a full bisection bandwidth network, where bandwidth is only bottlenecked at the access links of physical machines (which is also assumed in recent works [55, 42], *etc.*). Hence, the completion time of each flow is determined by the bandwidth allocated at the access links. Note that even if the assumption does not hold, our model still works with a minor change, by setting proper bandwidth capacities of physical machines.

Each application $k \in \mathcal{K} = \{1, 2, ..., K\}$ requires $m_k$ tasks, represented by $\mathcal{T}_k = \{1, 2, ..., m_k\}$. The $i$-th task of application $k$ is represented by $\mathcal{T}_k^i \in \mathcal{T}_k$. For simplicity, we assume that both of the computation stages consist of the same number (*i.e.*, $m_k/2$) of tasks. Given the type of the communication pattern and the number of tasks in each computation stage, we can obtain the network load matrix $\mathcal{D}_k$, where the $(i, j)$-th component $\mathcal{D}_k^{i,j}$ represents the amount of data to be sent by the flow between task $\mathcal{T}_k^i$

and $\mathcal{T}_k^j$. For example, if the total amount of intermediate data generated by application $k$ is $d_k$, an application with the shuffle pattern will have $\frac{d_k}{(m_k/2)^2}$ data to be sent between each task pair, while an application with the broadcast pattern will have $\frac{d_k}{m_k/2}$ data to be sent by each flow.

Let $r_k^{i,j}$ denote the bandwidth allocated to the $(i,j)$ communicating task pair of application $k$, then the completion time of the flow between the $(i,j)$ task pair is $\frac{\mathcal{D}_k^{i,j}}{r_k^{i,j}}$. The transfer time of an application is defined as the completion time of the slowest flow in the communication stage, which can be represented as $t_k = \max_{i,j,\mathcal{D}_k^{i,j}\neq 0} \frac{\mathcal{D}_k^{i,j}}{r_k^{i,j}}$ for application $k$.

As mentioned in Sec. 3.1, each application $k$ is associated with a weight $w_k$. The performance of application $k$ is expressed as $\frac{1}{t_k}$, which indicates that the shorter the transfer time, the better the performance. The fairness definition in Eq. (3.1) has the following equivalent form:

$$\frac{1}{t_k} = \frac{w_k}{\sum_k w_k} S, \quad \forall k \in \mathcal{K} \tag{3.2}$$

where $S$ is a positive variable called the *total performance-centric share*, which is upper bounded given the fixed amount of bandwidth capacity in the datacenter. Our objective is to fairly allocate bandwidth to achieve this upper bound, so that the performance achieved by each application is maximized. Substituting $t_k = \max_{i,j,\mathcal{D}_k^{i,j}\neq 0} \frac{\mathcal{D}_k^{i,j}}{r_k^{i,j}}$ yields

$$\min_{i,j,\mathcal{D}_k^{i,j}\neq 0} \frac{r_k^{i,j}}{\mathcal{D}_k^{i,j}} = \frac{w_k}{\sum_k w_k} S$$

Now we consider the link bandwidth capacity constraints on each server. Let the binary variable $X_{k,n}^i$ denote whether task $i$ of application $k$ is placed on server $n$, *i.e.*,

$$X_{k,n}^i = \begin{cases} 1, & \text{when } \mathcal{T}_k^i \text{ is placed on server } n \\ 0, & \text{otherwise} \end{cases}$$

The total egress rate of each task $\mathcal{T}_k^i$ placed on server $n$ is $\sum_{j,X_{k,n}^j=0} r_k^{i,j} \cdot X_{k,n}^i$. Note that if the task $\mathcal{T}_k^j$ receiving the intermediate data from $\mathcal{T}_k^i$ is also placed on server $n$, there will be no data sent through the network. Thus, we add the constraint of $X_{k,n}^j = 0$ in the summation. Summing over all tasks of an application placed on server $n$, and further summing over all the applications, we obtain the total egress rate of server $n$, which should not exceed the egress link capacity:

$$\textstyle\sum_k \sum_i \sum_{j,X_{k,n}^j=0} r_k^{i,j} \cdot X_{k,n}^i \le B_n^E$$

The same analysis applies to the ingress link of each server.

We are now ready to formulate the problem of maximizing performance while maintaining weighted performance-centric fairness:

$$\max_{\mathbf{r}} \quad S \tag{3.3}$$

$$\text{s.t.} \quad \min_{i,j,\mathcal{D}_k^{i,j}\neq 0} \frac{r_k^{i,j}}{\mathcal{D}_k^{i,j}} = \frac{w_k}{\sum_k w_k} S, \quad \forall k \in \mathcal{K} \tag{3.4}$$

$$\sum_k \sum_i \sum_{j,X_{k,n}^j=0} r_k^{i,j} \cdot X_{k,n}^i \le B_n^E, \quad \forall n \in \mathcal{N} \tag{3.5}$$

$$\sum_k \sum_j \sum_{i,X_{k,n}^i=0} r_k^{i,j} \cdot X_{k,n}^j \le B_n^I, \quad \forall n \in \mathcal{N} \tag{3.6}$$

where constraint (3.4) represents weighted performance-centric fairness, while constraints (3.5) and (3.6) correspond to the egress and ingress link capacity constraints at each machine.

Let $\alpha_k$ denote the performance of application $k$, *i.e.*, the reciprocal of its transfer time:

$$\alpha_k = \frac{1}{t_k} = \min_{i,j,\mathcal{D}_k^{i,j}\neq 0} \frac{r_k^{i,j}}{\mathcal{D}_k^{i,j}} \tag{3.7}$$

We can obtain the optimal value $S^*$ of the optimization problem (3.3)-(3.6) by solving the following optimization problem, which has the same value of $S^*$:

$$\max_{\boldsymbol{\alpha}} \quad S \tag{3.8}$$

$$\text{s.t.} \quad \alpha_k = \frac{w_k}{\sum_k w_k} S, \quad \forall k \in \mathcal{K} \tag{3.9}$$

$$\alpha_k = \frac{r_k^{i,j}}{\mathcal{D}_k^{i,j}}, \quad \forall i, j \in \mathcal{T}_k, \mathcal{D}_k^{i,j} \neq 0 \tag{3.10}$$

$$\sum_k \alpha_k \sum_i \sum_{j, X_{k,n}^j = 0} \mathcal{D}_k^{i,j} X_{k,n}^i \leq B_n^E, \forall n \in \mathcal{N} \tag{3.11}$$

$$\sum_k \alpha_k \sum_j \sum_{i, X_{k,n}^i = 0} \mathcal{D}_k^{i,j} X_{k,n}^j \leq B_n^I, \forall n \in \mathcal{N} \tag{3.12}$$

The intuition is that since the performance of each application is determined by the completion time of its slowest flow, it is efficient to make all the flows of an application finish at the same time, by allocating flows the amounts of bandwidth that have the same proportionality to their network load. In this way, no bandwidth is wasted in making some of the flows finish faster. Therefore, we can add constraint (3.10) without impacting the optimal $S^*$ of problem (3.3)-(3.6).

Replacing the variables of $\alpha_k$ with $S$ according to constraint (3.9), we transform problem (3.8)-(3.12) as follows:

$$\begin{aligned} \max \quad & S \\ \text{s.t.} \quad & S \cdot \textstyle\sum_k w_k^{'} b_{k,n}^E \leq B_n^E, \quad \forall n \in \mathcal{N} \\ & S \cdot \textstyle\sum_k w_k^{'} b_{k,n}^I \leq B_n^I, \quad \forall n \in \mathcal{N} \end{aligned}$$

where $w_k^{'} = \frac{w_k}{\sum_k w_k}$ represents the normalized weight of application $k$, and

$$b_{k,n}^E = \textstyle\sum_i \sum_{j, X_{k,n}^j = 0} \mathcal{D}_k^{i,j} X_{k,n}^i \tag{3.13}$$

$$b_{k,n}^I = \textstyle\sum_j \sum_{i, X_{k,n}^i = 0} \mathcal{D}_k^{i,j} X_{k,n}^j \tag{3.14}$$

representing the total amount of traffic generated by $k$ to be transmitted through the egress link at server $n$, and the total amount of data received by $k$ through the ingress link at server $n$, respectively.

The optimal solution is thus expressed as:

$$\begin{aligned} S^* = \min\{ \quad & \min_{n, \sum_k b^E_{k,n} \neq 0} \frac{B^E_n}{\sum_k w'_k b^E_{k,n}} \quad , \\ & \min_{n, \sum_k b^I_{k,n} \neq 0} \frac{B^I_n}{\sum_k w'_k b^I_{k,n}} \quad \} \end{aligned} \tag{3.15}$$

With the maximum total performance-centric share $S^*$, the optimal performance of each application is obtained as $\alpha^*_k = \frac{w_k}{\sum_k w_k} S^*$.

Finally, according to constraint (3.10), we derive the optimal rate allocation as:

$$r^{i,j*}_k = \frac{w_k}{\sum_k w_k} S^* \cdot \mathcal{D}^{i,j}_k$$

Note that in our original problem (3.3)-(3.6), according to Eq. (3.7), constraint (3.4) can be transformed as:

$$\begin{aligned} \alpha_k &= \frac{w_k}{\sum_k w_k} S, \quad \forall k \in \mathcal{K} \\ \alpha_k &\leq \frac{r^{i,j}_k}{\mathcal{D}^{i,j}_k}, \quad \forall i, j \in \mathcal{T}_k, \mathcal{D}^{i,j}_k \neq 0 \end{aligned} \tag{3.16}$$

where Eq. (3.16) indicates the flexibility to increase some of the $r^{i,j}_k$ as long as the capacity constraint is still maintained. However, this would not result in any improvement of the performance $\alpha_k$. Since the application performance is our main concern, we simply allocate the minimum amounts to achieve the specified performance, according to constraint (3.10).

## 3.3 Reconciling Fairness and Bandwidth Utilization

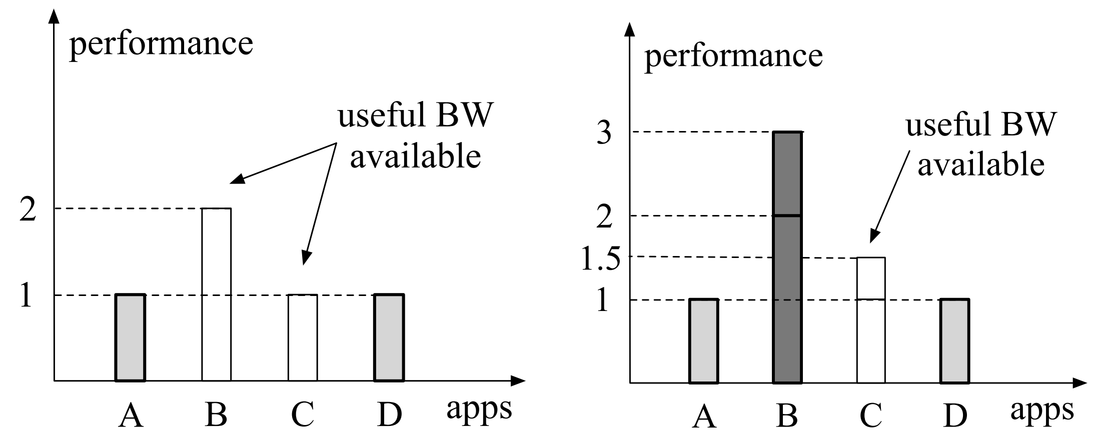
It is well known that tradeoff exists between fairness and efficiency. In this and the following sections, we will investigate how the bandwidth allocation following the strict definition of weighted performance-centric fairness results in a lack of efficiency, with interpretations from two perspectives. In correspondence, we will present two approaches to improve the efficiency.

The first interpretation of efficiency is from the perspective of bandwidth utilization. When the weighted performance-centric fairness is achieved, the residual bandwidth can be categorized into two classes:

- *useful bandwidth* — the link bandwidth that once allocated to an application, the performance of the application can be improved. It is a concept that is relative to applications, *i.e.*, the useful bandwidth to an application is not necessarily the useful bandwidth to another. For an application $k$, the useful bandwidth represents a set of available (non-zero) bandwidth on all the access links that $k$'s flows traverse.
- *useless bandwidth* — the link bandwidth that can not improve the performance of any application, *i.e.*, the link bandwidth that is not the *useful bandwidth* to any application.

The maximal efficiency is achieved when all the residual bandwidth is *useless bandwidth*.

It is intuitive that when the strict performance-centric fairness is enforced, maximal efficiency may not be achieved, because some applications are not allowed to utilize useful bandwidth for the purpose of maintaining performance proportionality. To eliminate such an inherent conflict, we define an extended version of weighted performance-centric fairness, which allows the residual bandwidth to be iteratively allocated to the applications that can improve their performance with the allocation, according to the performance proportionality regulated by the original weighted performance-centric fairness, until all



(a) 1st round: $A$ and $D$ no longer have *useful bandwidth*, and their allocations have been fixed.

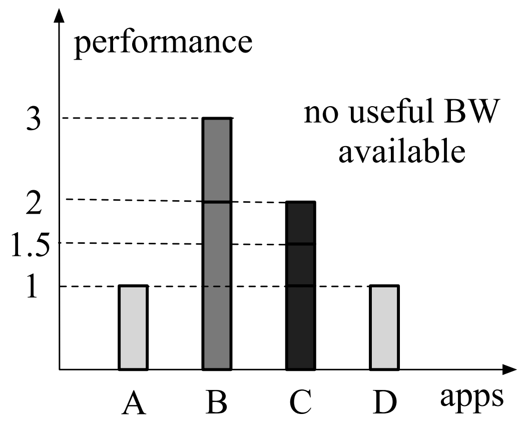
(b) 2nd round: $B$ no longer has *useful bandwidth*, and its allocation has been fixed.

Figure 3.3: An illustration of allocating bandwidth among applications $A$, $B$, $C$ and $D$ iteratively to achieve maximal efficiency, following the extended version of weighted performance-centric fairness. The weights of these four applications are 1, 2, 1, 1, respectively.

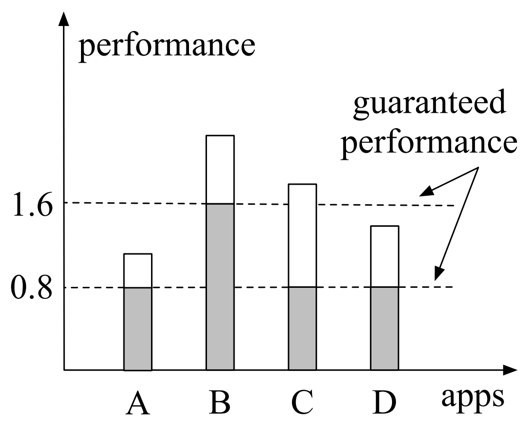
the residual bandwidth is the *useless bandwidth*. In this way, all the *useful bandwidth* (relative to some applications) will be utilized, and the efficiency regarding bandwidth utilization is maximized.

To better understand the idea, we present an illustrative example of the iterative allocation in Fig. 3.3 and Fig. 3.4. Suppose that according to the weighted performance-centric fairness as previously defined, the maximal total performance-centric share $S^*$ is calculated as 5, so that application $A$, $C$ and $D$ with the same weight of 1 obtains the performance of 1, and application $B$ with a weight of 2 achieves the performance of 2. However, there is still *useful bandwidth* to $B$ and $C$ that should be utilized to improve efficiency. Hence, in the first round, we simply fix the allocation of $A$ and $D$ to whom there is no longer *useful bandwidth*, as shown in Fig. 3.3a.

Then we remove $A$ and $D$ from our consideration, and allocate the available bandwidth among $B$ and $C$ in the second round, according to the weighted performance-centric fairness. Still, we compute $S^*$ for the problem where $B$ and $C$ are sharing the available bandwidth, based on which we can derive their performance as 3 and 1.5, proportional



(a) 3rd round: $C$ no longer has *useful bandwidth*, and its allocation has been fixed.



(b) Guaranteed performance and add-on performance achieved by $A$, $B$, $C$ and $D$.

Figure 3.4: An illustration of allocating bandwidth among applications $A$, $B$, $C$ and $D$ iteratively to achieve maximal efficiency, following the extended version of weighted performance-centric fairness. The weights of these four applications are 1, 2, 1, 1, respectively. (continued)

to their weights. As shown in Fig. 3.3b, we fix the allocation of $B$ in this round, while leaving the allocation of $C$ for the next round, since there is still *useful bandwidth* to $C$. Finally, $C$ will be allocated all the available *useful bandwidth* to it, to achieve a performance of 2, as shown in Fig. 3.4a. When the allocation completes, no *useful bandwidth* is left idle and the efficiency is maximized.

This allocation process can also be interpreted with progressive filling [56] that is used to achieve max-min fairness in traditional link bandwidth allocation. (Note that progressive filling is mentioned for the ease of illustrating the general idea. Though sharing the similar philosophy, our allocation algorithm to be elaborated later is carefully designed for the more complex context of the performance-centric fairness.) Starting with all the performance equal to 0, we increase the amounts of bandwidth to all the applications so that their performance increases at the rate proportional to their weights. When an application no longer has any *useful bandwidth*, its allocation is fixed. Then we continue increasing bandwidth for the remaining applications in a similar way until no *useful bandwidth* is available to any application.

From this perspective, we can see that in Fig. 3.3 and Fig. 3.4, the four applications

**Algorithm 1:** Bandwidth allocation to achieve relaxed performance-centric fairness and maximal utilization.

```
Input:
    Bandwidth capacity: $B_n^E, B_n^I, \forall n \in \mathcal{N}$;
    Network load matrix $\mathcal{D}_{i,j}^k$ and weight $w_k, \forall k \in \mathcal{K}$;
    Task placement across physical machines: $X_{k,n}^i$;
Output:
    Bandwidth allocation for all applications: $r_{i,j}^k$;
1: Initialize $\mathcal{N}^E = \mathcal{N}, \mathcal{N}^I = \mathcal{N}$;
2: while $\mathcal{K} \neq \varnothing$ do
3:     Calculate $w_k^{'} = w_k / \sum_{k \in \mathcal{K}} w_k$;
4:     Calculate $S^*$ according to Eq. (3.15);
5:     Obtain the application set $\mathcal{A}$, where each application has at least one saturated flow;
6:     Allocate bandwidth to all applications in $\mathcal{A}$ as:
           $$r_k^{i,j} = w_k^{'} S^* \mathcal{D}_k^{i,j};$$
7:     Update residual link bandwidth $B_n^E$ and $B_n^I$;
8:     Obtain saturated link sets $\mathcal{M}^E$ and $\mathcal{M}^I$;
9:     Update the link sets as:
       $\mathcal{N}^E = \mathcal{N}^E - \mathcal{M}^E; \mathcal{N}^I = \mathcal{N}^I - \mathcal{M}^I$;
10:    Update the application set as: $\mathcal{K} = \mathcal{K} - \mathcal{A}$;
11: end while
```

increases their performance with the ratio of their increasing rates as $1:2:1:1$, same to their weight proportionality. When the performance of $A$ increases to 1, $A$ and $D$ no longer have *useful bandwidth*, so that their performance is fixed as 1 in Fig. 3.3a. Then we continue increasing the allocation to $B$ and $C$, until $B$ exhausts all the *useful bandwidth*. Since their performance increasing rates are proportional to their weights, $C$ achieves the performance of 1.5 when $B$ is fixed at the performance of 3, as shown in Fig. 3.3b. Finally, we increase the allocation of the last application $C$ until all the *useful bandwidth* is used and $C$ achieves the performance of 2, as illustrated in Fig. 3.4a.

Our iterative allocation approach is summarized in Algorithm 1. We first compute the normalized weight of each application (`Line 3`) and the maximal total performance-centric fair share (`Line 4`) according Eq. (3.15) as: $S^* = \min\{\min_{n \in \mathcal{N}^E, \sum_{k \in \mathcal{K}} b_{k,n}^E \neq 0} \frac{B_n^E}{\sum_{k \in \mathcal{K}} w_k^{'} b_{k,n}^E}$ $\min_{n \in \mathcal{N}^I, \sum_{k \in \mathcal{K}} b_{k,n}^I \neq 0} \frac{B_n^I}{\sum_{k \in \mathcal{K}} w_k^{'} b_{k,n}^I}\}$, which is the analytic solution to the optimization problem (3.8)-(3.12) over the application set $\mathcal{K}$, the egress link set $\mathcal{N}^E$ and the ingress link set $\mathcal{N}^I$. Then we find all the applications that have at least one of their flows traversing a saturated link (`Line 5`), represented by the set $\mathcal{A} = \{k \in \mathcal{K} \mid \sum_k w_k^{'} b_{k,n}^E = B_n^E || \sum_k w_k^{'} b_{k,n}^I = B_n^I\}$, which also implies that there is no *useful bandwidth* to them. (Note that for such an application, there may be some bandwidth available for its flows that traverse non-saturated links. However, there is no *useful bandwidth* to this application, since it can not improve its performance, as long as one of its flows does not obtain any available bandwidth in a saturated link.) These applications in the set $\mathcal{A}$ are allocated with the amounts of bandwidth in `Line 6` to achieve their respective fair share of performance. The residual bandwidth in each link is then updated in `Line 7` by subtracting the amount that has been allocated to applications in $\mathcal{A}$, represented as follows:

$$B_n^E = B_n^E - S^* \sum_{k \in \mathcal{A}} w_k^{'} b_{k,n}^E, \ \forall n \in \mathcal{N}^E$$
$$B_n^I = B_n^I - S^* \sum_{k \in \mathcal{A}} w_k^{'} b_{k,n}^I, \ \forall n \in \mathcal{N}^I;$$

Finally, we obtain the saturated egress and ingress link sets as $\mathcal{M}^E = \{n \in \mathcal{N}^E \mid B_n^E = 0\}$, $\mathcal{M}^I = \{n \in \mathcal{N}^I \mid B_n^I = 0\}$, remove them from the link sets $\mathcal{N}^E$ and $\mathcal{N}^I$ (`Lines 8-9`), and remove all the applications in $\mathcal{A}$ from the application set $\mathcal{K}$ (`Line 10`).

As long as there are still applications that have not yet been allocated, *i.e.*, $\mathcal{K} \neq \varnothing$, we continue with a new round of allocation following the weighted performance-centric fairness, by solving the problem (3.8)-(3.12) over a reduced set of sharing applications and links. When the algorithm terminates, all the *useful bandwidth* have been utilized so that the maximal efficiency is achieved. Meanwhile, in each round, the weighted performance-centric fairness is achieved among the applications in $\mathcal{A}$, thus the extended version of weighted performance-centric fairness is satisfied by Algorithm 1.

Now we analyze the complexity of our Algorithm 1. In each iteration, `Line 4` calculates $S^*$ by looping through the application set $\mathcal{K}$ and the server set $\mathcal{N}$. Hence, the complexity of this step is $O(NK)$, where $N$ is the number of servers (where at least a flow is originated or destined) and $K$ is the number of concurrent applications. `Line 6` loops through all the flows of applications in $\mathcal{A}$ with a complexity of $O(F)$, where $F$

is the total number of concurrent flows across all the applications. The complexity of `Line 7` is $O(NK)$ and other steps are $O(N)$ or $O(K)$. Therefore, each iteration has the complexity of O(NK+F). With at most $K$ iterations, the time complexity of Algorithm 1 is $O((NK+F)K)$. As we observe, the complexity is linearly or at most quadratically increasing with an input, which is feasible and practical for bandwidth allocation in modern datacenters.

## 3.4 Trading Fairness for Social Welfare

Another interpretation of efficiency is from the perspective of the social welfare. Each application $k$ has a utility function, determined by its performance $\alpha_k$. For simplicity, we choose the log function of performance as the utility function for each application:

$$U_k(\alpha_k) = \log \alpha_k, \ \forall k \in \mathcal{K}$$

The *social welfare* is the total utility achieved by all the applications, *i.e.*, $\sum_{k \in \mathcal{K}} U_k(\alpha_k)$. The maximal efficiency is achieved when the *social welfare* across all the applications is maximized.

For this interpretation, there is still conflicts between the requirements of weighted performance-centric fairness and maximal social welfare. The underlying reason is twofold, which will be illustrated by two simple examples as follows, where application $A$ and $B$ are sharing the same set of links:

- To achieve 1 unit of performance, $A$ requires 1 unit of bandwidth, while $B$ requires 2 units. $A$ and $B$ have the same weights. According to the allocation that satisfies the weighted performance-centric fairness, $A$ and $B$ will achieve the same performance of 2 units, and the social welfare is $\log 4$. If we reduce 1 unit of bandwidth from $B$, and reallocate it to $A$, then the performance achieved by $A$ and $B$ is 3 and 1.5, respectively. With this allocation, the social welfare is computed as $\log 4.5$, greater than that achieved by weighted performance-centric allocation.

- To achieve 1 unit of performance, $A$ and $B$ both require 1 unit of bandwidth. The weights of $A$ and $B$ are 1 and 2, respectively. According to the allocation that satisfies the weighted performance-centric fairness, $A$ will achieve the performance of 2 units, while $B$ achieves 4 units. The social welfare is $\log 8$. If we reduce 1 unit of bandwidth from $B$, and reallocate it to $A$, then both $A$ and $B$ will achieve the performance of 3. With this allocation, the social welfare is computed as $\log 9$, greater than that achieved by weighted performance-centric allocation.

To strive for a balance between the conflicting requirements of fairness and efficiency, we set the principle of $\alpha_k \geq w_k^{'} S$ for bandwidth allocation, where $S \in (0, S^*]$ is considered as the degree of fairness relaxation that can be tuned to trade fairness for efficiency. This constraint ensures that, at a minimum, each application can be guaranteed a weighted fair share $w_k^{'} S$ with respect to its performance. A larger $S$ indicates a larger weighted fair share that each application will be guaranteed, while a smaller $S$ represents more relaxation on fairness.

As shown in Fig. 3.4b, the shaded areas represent the *guaranteed performance* of the four applications, which requires a guaranteed amount of bandwidth. The residual bandwidth, called the *flexible bandwidth*, will be allocated to maximize the social welfare without the concern of fairness, which results in the *add-on performance* represented by the white areas. In this example, $S$ is 4, so that the *guaranteed performance* of $A$ is $4 \cdot \frac{1}{5} = 0.8$, and that of $B$ is $4 \cdot \frac{2}{5} = 1.6$. If we decrease $S$, there will be more relaxation on fairness, as the *guaranteed performance* will be reduced. However, more *flexible bandwidth* will be available to be allocated for the purpose of improving the social welfare.

To maximize the overall social welfare of all applications with a certain degree of relaxation on performance-centric fairness, we formulate the following optimization problem:

$$\max_{\alpha} \quad \textstyle\sum_k U_k(\alpha_k) \tag{3.17}$$
$$\text{s.t.} \quad \alpha_k \geq w_k^{'} S, \quad \forall k \in \mathcal{K} \tag{3.18}$$
$$\textstyle\sum_k \alpha_k \cdot b_{k,n}^E \leq B_n^E, \quad \forall n \in \mathcal{N} \tag{3.19}$$
$$\textstyle\sum_k \alpha_k \cdot b_{k,n}^I \leq B_n^I, \quad \forall n \in \mathcal{N} \tag{3.20}$$

where $b_{k,n}^E$ and $b_{k,n}^I$ are expressed in Eq. (3.13) and Eq. (3.14). Constraint (3.18) follows the aforementioned principle, while (3.19) and (3.20) are the capacity constraints of the egress and ingress link at each server.

Changing $\max \sum_k U_k(\cdot)$ to $\min - \sum_k U_k(\cdot)$, we can transform the previous optimization problem (3.17)-(3.20) to the following:

$$\min_{\alpha} \quad -\textstyle\sum_k U_k(\alpha_k) \tag{3.21}$$
$$\text{s.t.} \quad Eq.\ (3.18), (3.19), (3.20) \tag{3.22}$$

Since $U_k(\alpha_k) = \log \alpha_k$ is strictly concave [57], the objective function of problem (3.21)-(3.22) is strictly convex. Moreover, all of the constraints are affine. Hence, problem (3.21)-(3.22) is a convex optimization problem [57].

Let $\lambda_k, \forall k \in \mathcal{K}$ denote the Lagrange multipliers associated with constraint (3.18), and $\mu_n^E, \mu_n^I, \forall n \in \mathcal{N}$ associated with capacity constraints (3.19) and (3.20) respectively. The Lagrangian of problem (3.21)-(3.22) is as follows:

$$\begin{aligned}\mathcal{L}(\alpha, \lambda, \mu^E, \mu^I) &= -\textstyle\sum_k \log \alpha_k + \sum_k \lambda_k (w_k^{'} S - \alpha_k) \\ + \quad &\textstyle\sum_n \mu_n^E (\sum_k \alpha_k b_{k,n}^E - B_n^E) + \sum_n \mu_n^I (\sum_k \alpha_k b_{k,n}^I - B_n^I)\end{aligned}$$

It is obvious that there exists $\alpha = (\alpha_1, ..., \alpha_K)$ in the relative interior of the intersection of domains of all constraint functions, *i.e.*, $\alpha = (\alpha_1, ..., \alpha_K)$ satisfies the constraints: $\alpha_k > w_k^{'} S, \forall k \in \mathcal{K}$, $\sum_k \alpha_k b_{k,n}^E < B_n^E, \forall n \in \mathcal{N}$ and $\sum_k \alpha_k b_{k,n}^I < B_n^I, \forall n \in \mathcal{N}$. Hence, Slater's condition [57] is satisfied. And since the optimization problem (3.21)-(3.22) is differentiable and convex, the Karush-Kuhn-Tucker (KKT) conditions [57] are both sufficient and necessary for the optimality. Thus, we can derive the optimal solution by applying the KKT conditions:

$$\nabla_{\alpha_k} \mathcal{L}(\alpha, \lambda, \mu^E, \mu^I) = 0, \ \forall k \in \mathcal{K} \Longleftrightarrow$$

$$-1/\alpha_k - \lambda_k + \textstyle\sum_n \mu_n^E b_{k,n}^E + \sum_n \mu_n^I b_{k,n}^I = 0, \ \forall k \in \mathcal{K}$$

and

$$\begin{cases} \lambda_k(w_k^{'} S - \alpha_k) = 0, \ \forall k \in \mathcal{K} \\ \mu_n^E(\sum_k \alpha_k b_{k,n}^E - B_n^E) = 0, \ \forall n \in \mathcal{N} \\ \mu_n^I(\sum_k \alpha_k b_{k,n}^I - B_n^I) = 0, \ \forall n \in \mathcal{N} \\ \lambda_k \geq 0, \ \forall k \in \mathcal{K} \\ \mu_n^E \geq 0, \ \mu_n^I \geq 0, \ \forall n \in \mathcal{N} \end{cases}$$

Analyzing the solution $(\alpha_k^*, \forall k \in \mathcal{K})$ of the equations above, we derive the following insights:

1) If $\alpha_k^* = w_k^{'} S$ for some $k$, we have

$$-\tfrac{1}{w_k^{'} S} - \lambda_k + \textstyle\sum_n \mu_n^E b_{k,n}^E + \sum_n \mu_n^I b_{k,n}^I = 0. \tag{3.23}$$

Since $\lambda_k \geq 0$ and $\frac{1}{w_k^{'} S} > 0$, to satisfy Eq. (3.23) we have $\sum_n \mu_n^E b_{k,n}^E + \sum_n \mu_n^I b_{k,n}^I > 0$. Hence, there exists a server $n \in \{n | X_{k,n}^i \neq 0, \exists i \in \mathcal{T}^k\}$ that satisfies $\mu_n^E \neq 0$ or $\mu_n^I \neq 0$, from which we have

$$\textstyle\sum_k \alpha_k^* b_{k,n}^E - B_n^E = 0 \ \text{ or } \ \sum_k \alpha_k^* b_{k,n}^I - B_n^I = 0.$$

This indicates that among all the servers hosting application $k$'s tasks, there is at least one of them that has no idle link to improve $k$'s performance. Thus, the performance of $k$ only achieves its minimum guaranteed share.

2) If $\alpha_k^* > w_k^{'}S$, then we have $\lambda_k = 0$, and

$$-\frac{1}{\alpha_k^*} + \sum_n \mu_n^E b_{k,n}^E + \sum_n \mu_n^I b_{k,n}^I = 0$$

Similar with the analysis when $\alpha_k^* = w_k^{'}S$, there exists such $n \in \{n | X_{k,n}^i \neq 0, \exists i \in \mathcal{T}^k\}$ that satisfies $\sum_k \alpha_k^* b_{k,n}^E - B_n^E = 0$ or $\sum_k \alpha_k^* b_{k,n}^i - B_n^I = 0$, which indicates that the bandwidth is bottlenecked at some servers where tasks of application $k$ are placed.

In this case, we can represent $\alpha_k^*$ as follows:

$$\alpha_k^* = \frac{1}{\sum_n \mu_n^E b_{k,n}^E + \sum_n \mu_n^I b_{k,n}^I}. \tag{3.24}$$

The optimal solution can be obtained by solving the equations of KKT conditions in a centralized manner. The centralized solver needs to maintain the network load matrices $\mathcal{D}_k$ with the dimension of $m_k$ for each application $k$, as well as the task placement state matrices $X_{k,n}^i$ $(i \in \mathcal{T}^k)$ for the entire datacenter. With $K + 2N$ constraints in the optimization problem, the computational complexity and storage overhead will increase significantly as the number of applications and physical machines scales up.

To overcome the limit of the centralized approach, we propose a distributed algorithm to allocate bandwidth at each server with less computation and less state information, which will be presented in the next section.

## 3.5 Distributed Bandwidth Allocation for Maximizing Social Welfare

In this section, we first prove that there is no duality gap between the dual and the primal problem, and then apply the dual based decomposition to design a distributed algorithm for bandwidth allocation that can be implemented at each physical machine in parallel.

### 3.5.1 Dual Based Decomposition

Let $p^*$ represent the optimal value of the primal optimization problem (3.21)-(3.22). Considering the general situation that $\alpha_k > w_k^{'}S$, we use $\mathcal{X} \subset \mathcal{R}^K$ to denote the solution space defined by constraint (3.18). The Lagrangian of the primal problem under the general case is defined as $\mathcal{L}(\cdot) : \mathcal{X} \times \mathcal{R}^N \times \mathcal{R}^N \rightarrow \mathcal{R}$.

$$\begin{aligned}\mathcal{L}(\alpha, \mu^E, \mu^I) = -\textstyle\sum_k \log \alpha_k + \sum_n \mu_n^E(\sum_k \alpha_k b_{k,n}^E - B_n^E) \\ +\textstyle\sum_n \mu_n^I(\sum_k \alpha_k b_{k,n}^I - B_n^I)\end{aligned} \tag{3.25}$$

where $\mu^E = (\mu_1^E, ..., \mu_N^E)$ and $\mu^I = (\mu_1^I, ..., \mu_N^I)$ are the dual variables associated with constraints (3.19, 3.20). The Lagrange dual function $g(\cdot) : \mathcal{R}^N \times \mathcal{R}^N \rightarrow \mathcal{R}$ is defined as the minimum value of the Lagrangian $\mathcal{L}(\alpha, \mu^E, \mu^I)$ over $\alpha$:

$$g(\mu^E, \mu^I) = \min_{\alpha} \mathcal{L}(\alpha, \mu^E, \mu^I) = \mathcal{L}(\alpha^*, \mu^E, \mu^I) \tag{3.26}$$

Thus, the dual problem of the optimization problem (3.21)-(3.22) is:

$$d^* = \max_{\mu^E \in \mathcal{R}_+^N, \mu^I \in \mathcal{R}_+^N} \qquad g(\mu^E, \mu^I) \tag{3.27}$$

It has been shown in the previous section that Slater's condition holds, which means that there exists a strictly feasible solution that satisfies strict inequality constraints. Based on Slater's theorem, the strong duality holds [57]. Therefore, there is no duality gap between $p^*$ and $d^*$, *i.e.*, there exists $\mu^{E*}, \mu^{I*}$ such that $d^* = g(\mu^{E*}, \mu^{I*}) = \mathcal{L}(\alpha^*, \mu^{E*}, \mu^{I*}) = p^*$.

To sum up, in order to obtain the optimal solution of the primal problem, we can solve the dual problem (3.27) that has zero duality gap. Substituting $g(\mu^E, \mu^I)$ using

Eq. (3.26) and Eq. (3.25), we have the following equivalent dual problem:

$$\begin{aligned}\max_{\mu_n^E \geq 0, \mu_n^I \geq 0} \quad & \min_{\alpha} \textstyle\sum_n \mu_n^E (\sum_k \alpha_k b_{k,n}^E - B_n^E) \\ + \quad & \textstyle\sum_n \mu_n^I (\sum_k \alpha_k b_{k,n}^I - B_n^I) - \sum_k \log \alpha_k\end{aligned}$$

As clearly observed, this problem is a maximization problem over variables $\mu_n^E$ and $\mu_n^I$, subject to the constraints that each variable is a positive real number. In the objective function, only the previous two terms are dependent upon the variables. Without coupling constraints, it can be further divided into $N$ subproblems as follows:

$$\begin{aligned}\max_{\mu_n^E, \mu_n^I} \quad & \min_{\alpha} \mu_n^E (\textstyle\sum_k \alpha_k b_{k,n}^E - B_n^E) + \mu_n^I (\sum_k \alpha_k b_{k,n}^I - B_n^I) \\ \text{s.t.} \quad & \mu_n^E \geq 0, \quad \mu_n^I \geq 0\end{aligned}$$

Each of the subproblem has two variables $\mu_n^E$ and $\mu_n^I$, which can be solved at the corresponding server $n$.

### 3.5.2 Distributed Algorithm Using Gradient Projection

We now design a distributed algorithm to solve the dual problem, thus the optimal solution $\alpha^*$ for the primal problem can be further derived. Following the previous efforts on the design of distributed bandwidth allocation ([58, 42], *etc.*), we choose gradient projection method, which is simple to apply as the projection is easily calculated for the dual problem with only non-negativity constraints. Our algorithm is proved to converge and is easy to be implemented at each physical machine, with a small overhead.

According to the gradient projection method, we update variable $\mu_n$ (representing both $\mu_n^E$ and $\mu_n^I$ for simplicity) iteratively at each server $n \in \{1, 2, ..., N\}$ and each $t \geq 0$

with the following recursion scheme:

$$\begin{aligned} \mu_n^{(t+1)} &= \left[\mu_n^{(t)} + \zeta \frac{\partial g}{\partial \mu_n}\right]^+, \\ &= \left[\mu_n^{(t)} + \zeta\Big( \sum_k \alpha_k b_{k,n} - B_n\Big)\right]^+ \end{aligned} \tag{3.28}$$

where $\zeta > 0$ denotes the step-size, $[x]^+ = \max\{0, x\}$, $\mu_n$ stands for $(\mu_n^E, \mu_n^I)$, and the same applies to $b_{k,n}$ and $B_n$.

**Theorem 1.** *Given $\mu^{(0)} \in \mathcal{R}_+^{2N}$ and $\zeta \in (0, 2/\tilde{K}]$, where $\tilde{K} = \sqrt{2N}\sum_n \sum_k \big(b_{k,n}^E + b_{k,n}^I\big) C_k \overline{\alpha}_k^2$ and $C_k = \max_n\{b_{k,n}^E, b_{k,n}^I\}$, the recursive sequence $\{\mu^{(t)}\}$ generated by Eq. (3.28) converges to the dual optimum $\mu^*$,* i.e., *$\lim_{t\to\infty} \mu^{(t)} = \mu^*$.*

*Proof.* We first prove that the gradient of the dual function $g(\mu)$ is $\tilde{K}$-Lipschitz continuous.

Let us define a function $\theta_k(\cdot)$ as

$$\theta_k(x) = 1/x, \quad x \geq 1/\overline{\alpha}_k,$$

where $\overline{\alpha}_k = \min\{\min_{n\in\mathcal{N}, b_{k,n}^E \neq 0} \frac{B_n^E}{b_{k,n}^E}, \min_{n\in\mathcal{N}, b_{k,n}^I \neq 0} \frac{B_n^I}{b_{k,n}^I}\}$, representing the upper bound for the performance of application $k$ (achieved when all the link bandwidth is allocated to $k$). The Lipschitz constant of $\theta_k(x)$ is easily obtained as $\overline{\alpha}_k^2$.

We also define $d_n^E(\mu)$ as the partial derivative of $g(\mu^E, \mu^I)$ with respect to $\mu_n^E$:

$$d_n^E(\mu) = \frac{\partial g(\mu^E, \mu^I)}{\partial \mu_n^E} = \sum_k \alpha_k^* b_{k,n}^E - B_n^E \tag{3.29}$$

Based on Eq. (3.24), we can represent $\alpha_k^*$ as a $\theta_k(\cdot)$ function:

$$\begin{aligned} \alpha_k^*(\mu) &= 1/(\textstyle\sum_n \mu_n^E b_{k,n}^E + \sum_n \mu_n^I b_{k,n}^I) \\ &= \theta_k(\textstyle\sum_n \mu_n^E b_{k,n}^E + \sum_n \mu_n^I b_{k,n}^I) \end{aligned}$$

Thus, $d_n^E(\mu)$ can be represented as:

$$d_n^E(\mu) = \sum_k b_{k,n}^E \theta_k(\sum_n \mu_n^E b_{k,n}^E + \sum_n \mu_n^I b_{k,n}^I) - B_n^E$$

Then we can derive the following:

$$\begin{aligned}
& |d_n^E(\mu) - d_n^E(\mu^{'})| \\
\leq \ & \sum_k b_{k,n}^E \big|\theta_k(\sum_n \mu_n^E b_{k,n}^E + \sum_n \mu_n^I b_{k,n}^I) \\
- \ & \theta_k(\sum_n \mu_n^{'E} b_{k,n}^E + \sum_n \mu_n^{'I} b_{k,n}^I)\big| \\
\leq \ & \sum_k b_{k,n}^E \overline{\alpha}_k^2 \sum_n (b_{k,n}^E |\mu_n^E - \mu_n^{'E}| + b_{k,n}^I |\mu_n^I - \mu_n^{'I}|) \\
\leq \ & \sum_k b_{k,n}^E \overline{\alpha}_k^2 C_k \sum_n (|\mu_n^E - \mu_n^{'E}| + |\mu_n^I - \mu_n^{'I}|) \\
= \ & \big(\sum_k b_{k,n}^E C_k \overline{\alpha}_k^2\big) \|\mu - \mu^{'}\|_1 \qquad (3.30)
\end{aligned}$$

where $\|\cdot\|_1$ is the $L_1$ norm in vector space, and $C_k = \max_n\{b_{k,n}^E, b_{k,n}^I\}$. Similarly, we have

$$|d_n^I(\mu) - d_n^I(\mu^{'})| \leq (\sum_k b_{k,n}^I C_k \overline{\alpha}_k^2) \|\mu - \mu^{'}\|_1 \qquad (3.31)$$

Summing up Eq. (3.30) and Eq. (3.31) over all $n \in \mathcal{N}$ yields:

$$\begin{aligned}
& \|d(\mu) - d(\mu^{'})\|_1 \\
= \ & \sum_n \Big(|d_n^E(\mu) - d_n^E(\mu^{'})| + |d_n^I(\mu) - d_n^I(\mu^{'})|\Big) \\
\leq \ & \Big(\sum_n \sum_k \big(b_{k,n}^E + b_{k,n}^I\big) C_k \overline{\alpha}_k^2\Big) \|\mu - \mu^{'}\|_1
\end{aligned}$$

For any $\mu \in \mathcal{R}_+^{2N}$, we have $\|\mu\| \leq \|\mu\|_1 \leq \sqrt{2N}\|\mu\|$ ($\|\cdot\|_2$ or $\|\cdot\|$ is the $L_2$ norm or Euclidean norm) in metric space. Thus, we have the following result:

$$\begin{aligned}
& \|d(\mu) - d(\mu^{'})\| \leq \|d(\mu) - d(\mu^{'})\|_1 \\
\leq \ & \Big(\sum_n \sum_k \big(b_{k,n}^E + b_{k,n}^I\big) C_k \overline{\alpha}_k^2\Big) \|\mu - \mu^{'}\|_1 \\
\leq \ & \Big(\sqrt{2N} \sum_n \sum_k \big(b_{k,n}^E + b_{k,n}^I\big) C_k \overline{\alpha}_k^2\Big) \|\mu - \mu^{'}\|
\end{aligned}$$

**Algorithm 2:** Distributed bandwidth allocation to achieve maximal social welfare, with a tunable degree of relaxation on performance-centric fairness.

```
Input:
    Bandwidth capacity: $B_n^E, B_n^I, \ \forall n \in \mathcal{N}$;
    Network load matrix $\mathcal{D}_{i,j}^k$ and weight $w_k, \ \forall k \in \mathcal{K}$;
    Tunable relaxation on fairness: $S \in (0, S^*]$;
    Task placement across physical machines: $X_{k,n}^i$;
    Iteration times: $T$; Step-size: $\zeta \in (0, 2/\tilde{K}]$;
Output:
    Bandwidth allocation for all applications: $r_{i,j}^k$;
 1: Initialize $\alpha_k = w_k^{'} S, \ \forall k \in \mathcal{K}$;
 2: Calculate $b_{k,n}^E$ and $b_{k,n}^I$ based on Eq. (3.13) and (3.14);
 3: while iterations $< T$ do
 4:   for all physical machine $n$ do
 5:     $\mu_n^E = \max(0, \mu_n^E + \zeta(\sum_k \alpha_k b_{k,n}^E - B_n^E))$;
 6:     $\mu_n^I = \max(0, \mu_n^I + \zeta(\sum_k \alpha_k b_{k,n}^I - B_n^I))$;
 7:   end for
 8:   for all $\alpha_k$ do
 9:     $\alpha_k = \frac{1}{\sum_n \mu_n^E b_{k,n}^E + \sum_n \mu_n^I b_{k,n}^I}$;
10:     if $\alpha_k < w_k^{'} S$ then
11:       $\alpha_k = w_k^{'} S$;
12:     end if
13:   end for
14:   iterations $++$;
15: end while
16: for all $r_k^{i,j}$ do
17:   $r_k^{i,j} = \alpha_k \cdot \mathcal{D}_k^{i,j}$;
18: end for
```

Therefore, the Lipschitz constant of $\nabla g(\mu)$ is $\tilde{K} = \sqrt{2N} \sum_n \sum_k \left(b_{k,n}^E + b_{k,n}^I\right) C_k \overline{\alpha}_k^2$. According to the proof in [58], if $\nabla g(\mu)$ is $\tilde{K}$-Lipschitz continuous, then given a step-size $\zeta \in (0, 2/\tilde{K}]$, $\mu^{(t)}$ will converge in $\mu^*$ as $t \to \infty$. □

Since there is no duality gap between the dual problem and the primal problem, $\alpha(\mu^{(t)})$ associated with $\mu$ converges to the primal optimum, *i.e.*,

$$\lim_{t \to \infty} \alpha(\mu^{(t)}) = \alpha^*$$

With the theoretical guidelines so far, we are able to design Algorithm 2 for distributed bandwidth allocation, which can be implemented at each physical machine in parallel. The only required state information to be maintained on each machine is the weights and network load matrices of the applications that have their tasks placed on this machine, which is easily obtained. For example, in MapReduce, the job tracker can monitor the progress of all the tasks, thus have an overview of the volumes of traffic to be transferred among tasks.

In each iteration, the dual variables $\mu_n^E$ and $\mu_n^I$ will be updated at each machine $n$ following Eq. (3.28), given the step-size $\zeta$ bounded by the global constant $2/\tilde{K}$ and the performance of each application $k$ that has its tasks placed on this machine, *i.e.*, $b_{k,n}^E \neq 0$ or $b_{k,n}^I \neq 0$. Given the updated dual variables, application performance $\alpha_k$ can be updated in the application manager, according to Eq. (3.24). Note that $b_{k,n}^E$ and $b_{k,n}^I$ are 0 if none of the tasks of application $k$ is placed on machine $n$. Hence, computing $\alpha_k$ only requires $\mu_n^E$ and $\mu_n^I$ from those machines where at least one task of application $k$ is placed, which incurs a small communication overhead.

If the gradient in Eq. (3.29) is negative, which means that there is residual egress bandwidth on machine $n$, then $\mu_n^E$ will decrease and $\alpha_k$ will increase, indicating that the idle bandwidth will be utilized to increase application performance. If the computed performance is beyond its lower bound, it will be set as the bound to restrict the allocation in the feasible sets. Finally, when the dual variables converge to the optimum, the rates of the flows on server $n$ can be easily computed and allocated.

## 3.6 Performance Evaluation

In this section, we first evaluate the overall application performance and the total utilities achieved by our bandwidth allocation algorithms with the guidance of weighted performance-centric fairness (either the extended version of fairness focusing on the bandwidth utilization or the tunable relaxation on fairness targeting the social welfare), compared with traditional per-flow fairness. Then we investigate how our second algorithm performs in tuning the tradeoff between maximizing the total utility and maintaining performance-centric fairness. We further evaluate the performance of the distributed algorithm with respect to the convergence. Finally, we discuss the practical concern of our algorithms.

### 3.6.1 Comparing Different Versions of Fairness

We first simulate a small private datacenter with 20 homogeneous physical machines, with 10 GB/s egress and ingress links. Without loss of generality, there are 6 data parallel applications running concurrently, each with 6 tasks randomly placed across the datacenter. The weights of these applications, referred to as $A$, $B$, $C$, $D$, $E$ and $F$ for convenience, are set as 1, 2, 3, 4, 5, 6, respectively. For each application, half of their tasks are transferring the intermediate data to another half, with the traffic volumes randomly chosen in the range of $[200, 300]$ MB. Each machine is able to shape the bandwidth of its flows.

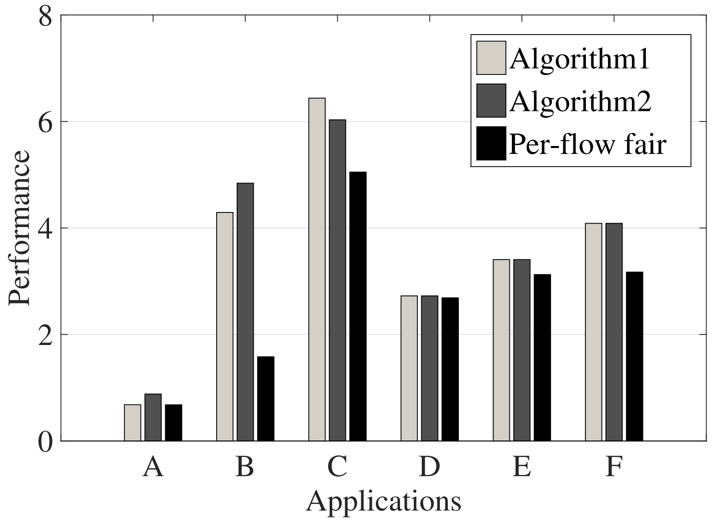


Figure 3.5: Performance of applications $A$, $B$, $C$, $D$, $E$ and $F$ achieved by Algorithm 1, Algorithm 2 and *Per-flow fair*, in a small datacenter with 20 servers. The weights are 1, 2, 3, 4, 5 and 6, respectively.

Given a problem setting resulted from a randomization, we realize our two algorithms respectively. For Algorithm 2, the tunable relaxation on fairness $S$ is set as the maximal value, which represents no relaxation. For comparison, we also apply the traditional flow-level max-min fair bandwidth allocation (referred to as *Per-flow fair*) used in TCP to the same setting. The performance of each application, measured as the reciprocal of its network transfer time as in Eq. (3.7), achieved by the three algorithms is illustrated in Fig. 3.5.

It is obvious that Algorithm 1 and Algorithm 2 outperform *Per-flow fair* to a large extent, both with up to $1.4X$ performance improvement. The reason is that weighted performance-centric fairness is aware of the characteristic of data parallel applications, so that any amount of bandwidth allocated among applications contributes to the performance gain. In contrast, with *Per-flow fair* allocation that is application agnostic, although some flows of an application may get a large amount of bandwidth to finish quickly, the application performance is determined by the slowest flow.

The application performance achieved by Algorithm 1 and Algorithm 2 is slightly different, because of the different allocation of the residual bandwidth after guaranteeing maximal weighted performance-centric fair share for all the applications. Algorithm 1 iteratively finds the applications that can still improve their performance with more bandwidth, and allocates available bandwidth to them so that the improvement of their performance is weight proportional. Algorithm 2, in contrast, allocates the residual bandwidth so as to maximize the total utility of all the applications. Note that in this particular case, the performance of $D$, $E$ and $F$ is exactly their respective fair share, with the same proportionality to their weights, while $A$, $B$ and $C$ are allocated the residual bandwidth by the two algorithms to improve efficiency, so that they achieve better performance beyond their original share. It is also worth noting that $D$ achieves almost the same performance with the three algorithms. The reason is that with different allocation from a different algorithm, the original bottlenecked flow may be accelerated, but another flow may become the bottleneck as its sharing flows are allocated with more bandwidth. If the new bottlenecked flow has the same completion time with the original one, the application performance will remain the same, which explains the performance of $D$.

With respect to the total utility, Algorithm 2 achieves 9.9305, larger than those achieved by Algorithm 1 and *Per-flow fair*, which are 9.4796 and 7.1747, respectively. Since the fairness relaxation parameter $S$ is set as the maximal value, the utility maximization in Algorithm 2 is restricted by the fairness constraint to the largest extent. If we reduce $S$ to have more relaxation on fairness, the total utility will be further increased.

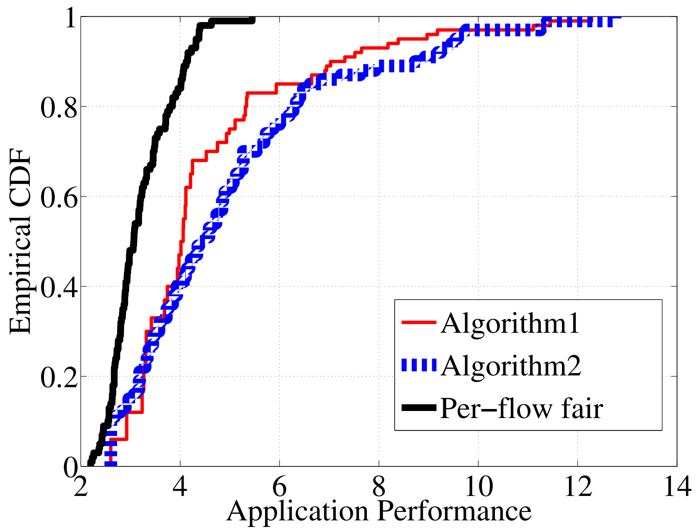


Figure 3.6: CDFs of the performance of 100 applications achieved by Algorithm 1, Algorithm 2 and *Per-flow fair*, in a large datacenter with 100 servers. The weights are the same.

Next, we extend our evaluation to a larger scale, with the number of machines increased to 100, and the number of applications increased to 100. For simplicity, all the applications have the same weight, and the fairness relaxation parameter $S$ is set as its maximal value. Fig. 3.6 presents the empirical CDFs of the application performance of the 100 applications achieved with three different algorithms. As is shown, all the application performance achieved with *Per-flow fair* is no larger than 5.5, while 17% of applications with Algorithm 1 and 30% of applications with Algorithm 2 achieve no smaller than 5.5, respectively. Clearly, the application performance achieved by our algorithms is much better than that achieved by *Per-flow fair*. Moreover, with respect to the total utilities, which are 209.4812, 219.7600 and 165.6202, respectively, our algorithms also outperform *Per-flow fair*.

### 3.6.2 Investigating Tunable Relaxation on Fairness

We continue to investigate the properties and evaluate the performance of Algorithm 2, with a detailed analysis in two typical scenarios shown in Fig. 3.7.

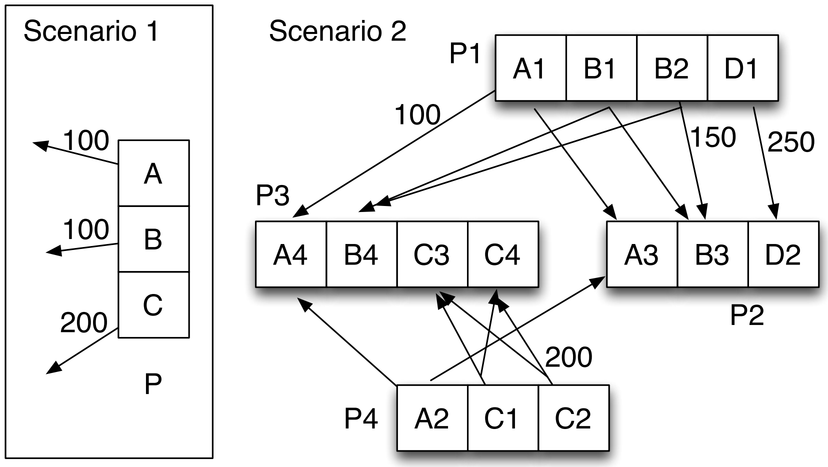


Figure 3.7: Two scenarios for evaluating the proposed bandwidth allocation.

In Scenario 1, three applications $A$, $B$ and $C$, with weights of 2, 1 and 1, encounter the same bottleneck at physical machine $P$, where their tasks have 100 MB, 100 MB and 200 MB intermediate data to be transferred, respectively. The link (both ingress and egress) bandwidth capacity of $P$ is 1 GB/s. Based on Eq. (3.15), the maximum total performance-centric share ($S^*$) of all the applications is computed as 8.

As shown in Fig. 3.8, as we relax the performance-centric fairness, *i.e.*, as we reduce $S$, the total utility of all the applications increases, indicating that the bandwidth becomes more efficiently utilized. Fig. 3.9 delves into several points of the tradeoff curve to show the performance achieved by these applications.

When $S$ is at its maximum as 8, the performance of $A$, $B$ and $C$ is 4, 2, 2 respectively, proportional to their weights, while the total utility is the lowest in the tradeoff curve. When we reduce $S$, the amount of bandwidth required to guarantee the total performance-centric share is reduced. The residual bandwidth can thus be freely allocated to the application whose utility will increase the most given this amount of

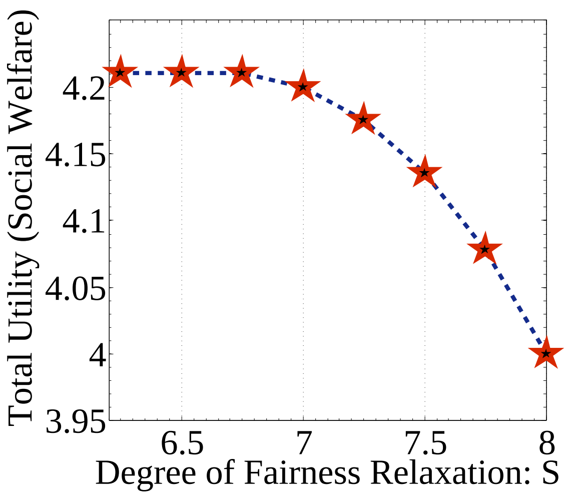


Figure 3.8: The tradeoff between the total performance-centric fair share and the total utility in scenario 1.

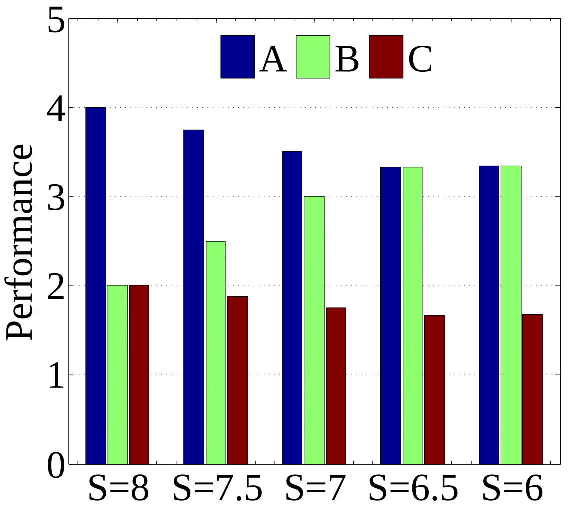


Figure 3.9: Performance of application $A$, $B$ and $C$ when $S$ is tuned at different values in scenario 1.

bandwidth. In this scenario, $B$ is the most competitive in grabbing the free bandwidth. It has fewer data to be sent compared with $C$ and its current performance is lower than $A$, so that its performance improvement will make the total utility increase the most.

This analysis is supported by Fig. 3.9. While all the applications are guaranteed the minimum performance-centric share, which is decreasing as we reduce $S$, $B$ can achieve higher performance than its minimum share by grabbing the free bandwidth. For example, when $S = 7$, the minimum performance-centric share is $\frac{2}{2+1+1} \cdot 7 = 3.5$ for $A$, and $\frac{1}{2+1+1} \cdot 7 = 1.75$ for $B$ and $C$. As shown in Fig. 3.9, $A$ and $C$ achieve the performance of 3.5 and 1.75 respectively, while $B$ achieves 3, which is more than its minimum share. When $S$ decreases to 6.5, the total utility will achieve its maximum. It will not increase any further with $S$ decreasing to 6 or below. Correspondingly, the application performance does not change when $S$ decreases below 6.5 according to our results (we only show $S = 6.5$ and $S = 6$ as examples in the figure).

Now we consider Scenario 2 as shown in Fig. 3.7, where four applications $A$, $B$, $C$ and $D$ with the same weight have their tasks placed across 4 servers, each with the bandwidth capacity of 3 GB/s. The bottleneck link encountered by $A$, $B$ and $C$ is at $P3$, while the

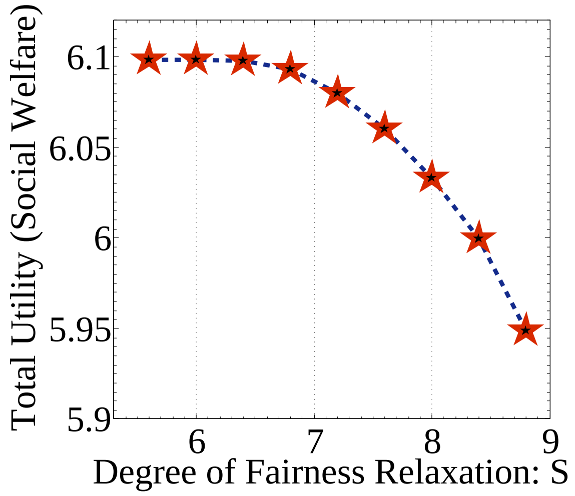


Figure 3.10: The tradeoff between the total performance-centric fair share and the total utility in scenario 2.

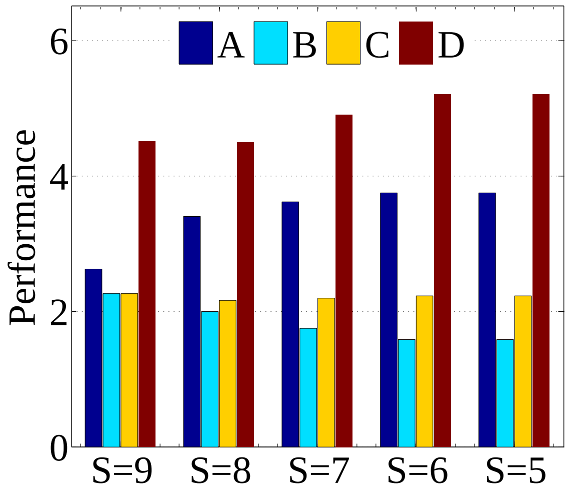


Figure 3.11: Performance of application $A$, $B$ $C$ and $D$ when $S$ is tuned at different values in scenario 2.

bottleneck of $D$ is $P1$. If we strictly follow performance-centric fairness, $S^*$ is computed as $\frac{120}{13}$, and $D$ will not be allowed to use the idle bandwidth on server $P1$ and $P2$. With the relaxation on fairness by reducing $S$, the idle bandwidth will be utilized, and there will be residual bandwidth after guaranteeing the minimum share to all the applications. The residual bandwidth can thus be allocated among applications to improve the total utility the most. Such a tradeoff between the total utility and the degree of fairness relaxation is verified in Fig. 3.10.

Fig. 3.11 illustrates the application performance achieved at different degrees of relaxation on performance-centric fairness. As we tune $S$, $B$ always achieves no more than its minimum guaranteed performance-centric share, while other applications achieve higher performance than their minimum shares. This can be explained by $B$'s heavy network load, which results in a smaller amount of utility increase compared with other applications, given the same amount of bandwidth allocation. Hence, the free bandwidth at $P1$ will be allocated to $D$, and the free bandwidth at $P3$ will be allocated to $A$, to improve the overall utility.

### 3.6.3 Convergence

Finally, we evaluate the convergence of Algorithm 2 in Scenario 1. As shown in Fig. 3.12, the dash lines and solid lines represent the performance of applications at different values of $S$, respectively. We choose the step-size according to Theorem 1 to guarantee the convergence. The performance of all the applications converges within about 70 iterations. To further evaluate our algorithm at a much larger scale, we simulate a datacenter with 100 physical machines hosting 100 data parallel applications, each of which has 4 tasks placed on different machines. Tasks of applications have different amounts (randomly chosen between 100 MB to 200 MB for simplicity) of data to be transferred in their communication stages. Fig. 3.13 shows the performance change of two randomly selected applications with an increasing number of iterations. We can see that the application performance is able to converge within 800 iterations.

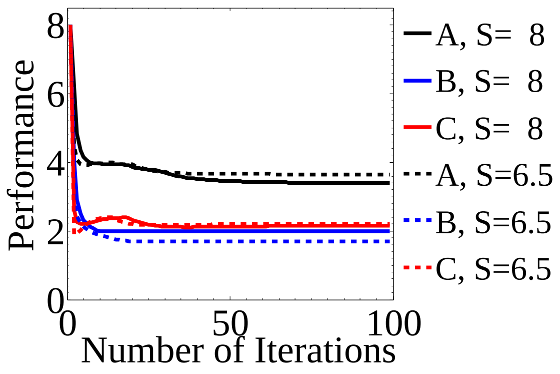


Figure 3.12: Convergence of performance of $A$, $B$ and $C$ when $S$ is 8 and 6, respectively, in Scenario 1.

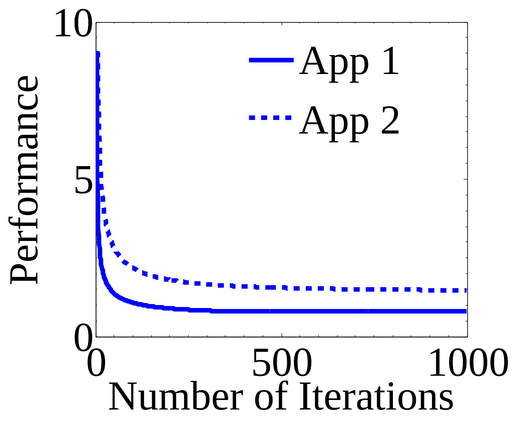


Figure 3.13: Convergence of performance of two applications in a large scale private datacenter.

### 3.6.4 Discussion

Algorithm 1 can be solved in polynomial time, as analyzed in Sec. 3.3, which is feasible in practice and fast especially in modern datacenters with powerful computation engine. Implementing this algorithm requires coordinating network transfers based on application demand, which can be realized with Software-Defined Network (SDN) [18]. The application demand can either be conveyed from application managers [59, 60], or anticipated by the SDN controller [61]. To be particular, the network load matrices of all the sharing flows should be obtained as input to Algorithm 1. With Apache Spark [2] as an example, the size and location of the output data from each `map` task can be obtained from the `MapOutputTracker` module. Once the placement of `reduce` tasks has been scheduled, the network load matrices can be easily obtained and conveyed to the SDN controller. As application weights and bandwidth capacities are also available, Algorithm 1 will run in the controller to calculate the bandwidth allocation, which is further enforced by setting maximal rates for switch queues and forwarding packets correspondingly [43].

Similarly, SDN can be applied to enforce the calculated bandwidth allocation by Algorithm 2. Specifically, the calculated rates at each server should be conveyed to the SDN controller, which then set up rate-limiting queues and forward packets accordingly. Moreover, a protocol is required for the communication between application managers and servers, to update the parameters when running the algorithm.

## 3.7 Concluding Remarks

Our focus in this chapter is to study the sharing of link bandwidth among applications running data parallel frameworks in a private datacenter. With the guideline that performance achieved by applications should be proportional to their weights, we propose a rigorous definition of performance-centric fairness and investigate the tradeoff between efficiency and fairness. From two perspectives of interpreting the efficiency, we design two algorithms to arbitrate the efficiency-fairness tradeoff. The first algorithm gradually improves the bandwidth utilization by introducing a relaxed version of fairness. With respect to the second interpretation of efficiency, a distributed algorithm is designed by solving a utility-maximization problem, constrained by a tunable degree of relaxation on performance-centric fairness. Such an algorithm can be implemented on each physical machine in a lightweight fashion. Our extensive simulations have shown that both algorithms have effectively improved efficiency with some relaxation on fairness, which result in up to $1.4X$ better application performance compared with the traditional fairness. The distributed algorithm is demonstrated to provide the flexibility to balance the tradeoff between the total utility and the degree of fairness relaxation.

# Chapter 4

# Utility-Optimal Coflow Scheduling

In this chapter, we further study the allocation of network resources among sharing jobs that better satisfy different job requirements. Due to their inherent nature, different jobs have widely diverging requirements with respect to their completion times: an interactive query in a web application should not be similarly treated as a background job for data analytics. Intuitively, different jobs have different sensitivity to their completion times, which can be characterized by different utility functions. With this observation, we argue that more time-sensitive coflows (from time-sensitive jobs) should be allocated more network resources, allowing them to complete earlier, achieving a higher utility based on their utility functions. This differentiates our work from the existing research efforts on coflow scheduling aimed at minimizing coflow completion times and meeting coflow deadlines, which treat coflows identically as equal citizens.

In what follows, we present our design and implementation of a new utility optimal scheduler to provide differential treatment to coflows with different degrees of sensitivity to completion times, yet still satisfying max-min fairness across these coflows. The scheduling algorithm is designed based on the theoretical study of a lexicographical optimization problem. We decouple the problem into subproblems with integer variables, which is rigorously proved to be equivalent to linear programming problems that can be efficiently solved.

## 4.1 Background and Motivation

### 4.1.1 Completion-Time-Dependent Utility

In a shared datacenter cluster, different jobs may have different sensitivity to their completion times. The diverging range of sensitivity can be captured by different utility functions that are dependent upon completion times [6]. As shown in Fig. 4.1, the two solid lines represent the utility functions for completion-time-critical jobs, which decrease sharply to zero after the target completion time. On the contrary, the utility value of a completion-time-insensitive job keeps the same along with time, represented by the horizontal line with star markers. In between, the two dashed lines decrease gradually with time, which characterize utilities of jobs that are completion-time-sensitive.

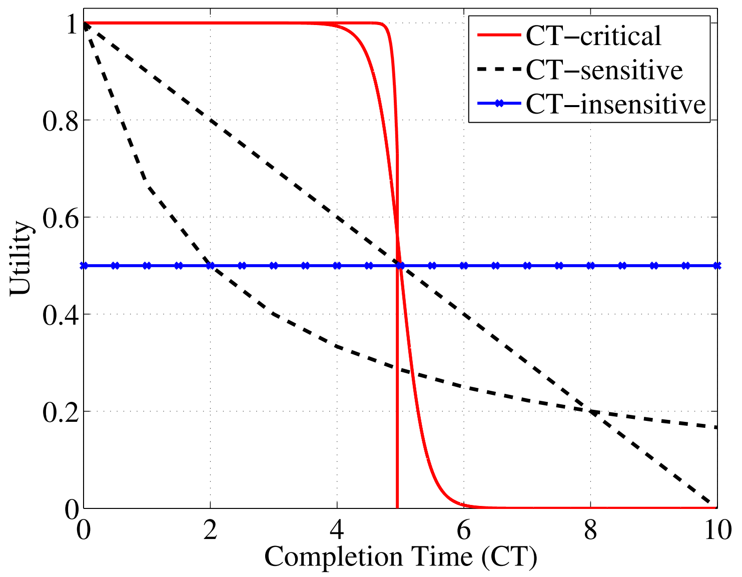


Figure 4.1: Completion-time-dependent utility functions.

The completion time of a job depends on both the computation time and the coflow completion time, *i.e.*, the completion time of the slowest flow. As the compution time is predictable due to mature techniques in allocation and isolation of computing resources, the job utility function can be easily translated to the coflow-completion-time dependent utility function. For convenience, the value of this function is referred to as the *coflow* *utility*, which is to be optimized in our network resource allocation.

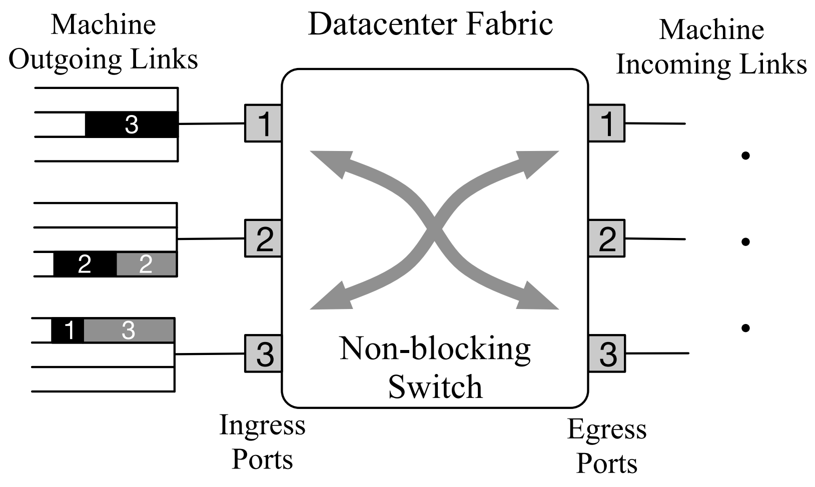


Figure 4.2: Coflows to be scheduled through a 3 × 3 datacenter fabric.

### 4.1.2 Network Shared by Coflows

As a common practice, we consider a congestion-free datacenter network, which can be abstracted as a giant non-blocking switch (*e.g.*, [7, 62, 4, 8]) that interconnects all the machines. This is reasonable and practical, thanks to the recent efforts in full bisection bandwidth topologies (*e.g.*, [54, 63]). In such a switch model, as illustrated in Fig. 4.2, each ingress port corresponds to the outgoing link of the connected machine, where the data is transferred from the machine to the network, and each egress port represents the incoming link of the connected machine which receives data from the network.

In the example shown in Fig. 4.2, 3 machines are interconnected by a non-blocking switch. Coflow $C1$ and $C2$ arrive at time 0, of which the flows are represented by the black and grey blocks in the figure, respectively. $C1$ has 3 flows transferring 3, 2 and 1 units of data; $C2$ has 2 flows with sizes of 2 and 3. The virtual input queues at the ingress ports are used for convenience to illustrate the source and destination of these flows. For example, at machine 1, a flow of $C1$ transfers 3 units of data to machine 2; at machine 2, flows of $C1$ and $C2$ each transfer 2 units of data to machine 3.

### 4.1.3 Utility Optimal Scheduling

The typical objective of existing coflow scheduling mechanisms was to minimize the average coflow completion time (CCT) or to meet coflow deadlines, and they did not account for different sensitivity levels to completion times. To better satisfy job requirements, we wish to optimize utilities achieved across all the competing coflows to achieve max-min fairness.

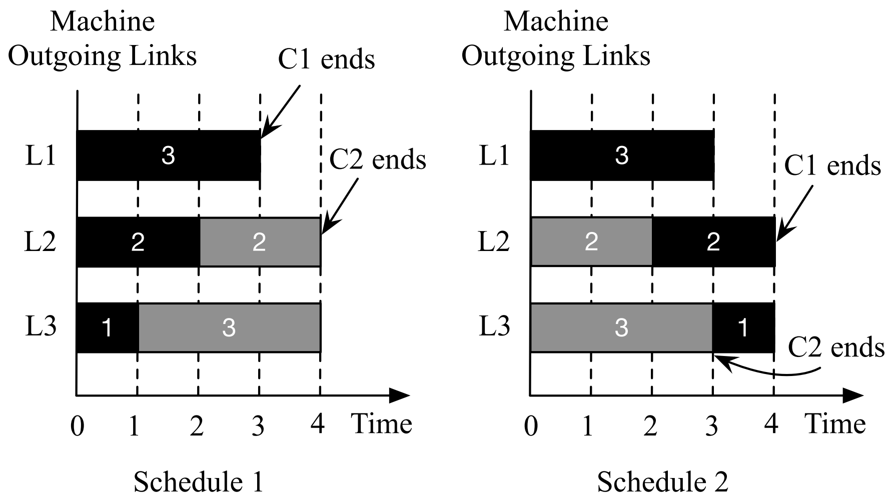
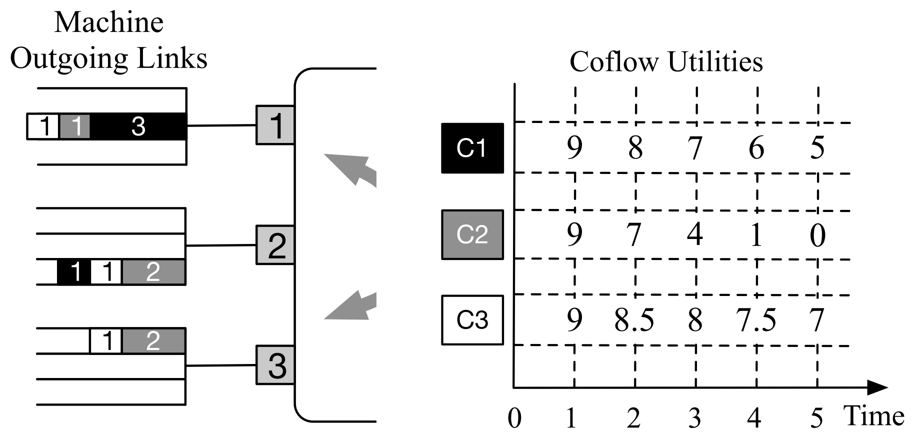


Figure 4.3: Two possible schedules for coflows $C1$ and $C2$. They are the same with respect to the average coflow completion time, but they achieve different coflow utilities.

Fig. 4.3 illustrates two schedules of coflows $C1$ and $C2$ aforementioned. Assume that $C1$ is completion-time-insensitive, which achieves a constant utility as long as it completes, while $C2$ is completion-time-sensitive, with its utility decreasing gradually with increasing completion times. A scheduler that tries to minimize the average CCT would choose either of these schedules, as both of them are optimal, making two coflows complete in 3 and 4 time units. However, when coflow utilities are to be optimized, Schedule 2 is the only optimal one, because it outperforms Schedule 1 with the same utility for $C1$ and better utility for $C2$. In this way, the awareness of coflow utility results in the optimal schedule that best satisfies coflow requirements.

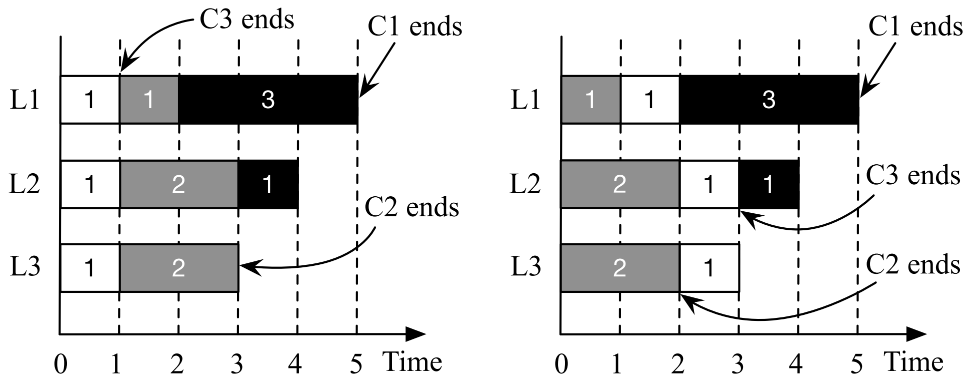
Further, we consider a more specific example shown in Fig. 4.4 for utility optimal



(a) Input (b) Coflow utilities

Figure 4.4: Three coflows to be scheduled.

schedule among coflows $C1$, $C2$ and $C3$. As shown in Fig. 4.4a, the 2 flows of $C1$, corresponding to the black blocks, transfer 3 and 1 units of data respectively; $C2$ has 3 flows (grey) transferring 1, 2 and 2 units of data; $C3$ has 3 flows (white) transferring 1 unit of data each. The utilities of coflows achieved at different completion times are presented in Fig. 4.4b, where we can see that $C2$ is the most sensitive to its completion times, while $C3$ is the least sensitive.



(a) Schedule to minimize CCT (b) Schedule to maximize utilities

Figure 4.5: Utility optimal schedule among three coflows.

Fig. 4.5a and Fig. 4.5b, respectively, present the schedule that minimizes the average CCT, and the schedule that optimizes coflow utilities with max-min fairness. Fig. 4.5a achieves 5, 3 and 1 units of coflow completion times, better than Fig. 4.5b with 5, 2 and 3 when the average CCT is considered. However, with respect to coflow utilities, Fig. 4.5a obtains the utility values of 5, 4 and 9, while Fig. 4.5b achieves 5, 7 and 8. Clearly, the schedule in Fig. 4.5b is utility optimal, with max-min fairness achieved among coflow utilities.

Note that for simplicity of illustration, there is no contention on machine incoming links in these examples. In general, however, both outgoing and incoming links may experience contention, and the utility optimal schedule can be more effective.

## 4.2 Model and Formulation

We consider a cloud cluster with a set of servers denoted by $\mathcal{N} = \{1, 2, \cdots, N\}$, which are shared by multiple data parallel jobs. A set of coflows $\mathcal{K} = \{1, 2, \cdots, K\}$ are submitted by these concurrently running jobs, to transfer the intermediate data through the network in their communication stages.

As mentioned in the previous section, the network is considered as a non-blocking switch, thus our coflow schedule takes place at its ingress and egress ports, corresponding to the incoming and outgoing links at each server. For simplicity, we assume that at each time slot $t \in \mathcal{T} = \{1, 2, \cdots, T\}$, each server can transmit one unit of data through its outgoing (egress) link, and receive one unit of data through its incoming (ingress) link.

A flow that belongs to coflow $k \in \mathcal{K}$ is represented by $D^k_{i,j},\ i, j \in \mathcal{N}$, which specifies that it transfers $D^k_{i,j}$ units of data from server $i$ to $j$. Coflow $k$ completes when its last flow finishes at time $t \in \mathcal{T}$, and achieves a utility of $u^t_k$ which is determined by its non-increasing utility function $\mathcal{U}_k(t)$.

As server links are shared by flows from multiple coflows, we would like to obtain a utility optimal coflow schedule without exceeding link capacities. To be particular, our objective is to maximize the worst utility achieved among all the coflows, then maximize the next worst utility without impacting the previous one, which is executed repeatedly until utilities have been optimized for all the coflows. Such an objective can be rigorously formulated as a *lexicographical maximization* problem, with the following definitions as the basis.

**Definition 1.** *Let $\langle \boldsymbol{u} \rangle_k$ denote the k-th ($1 \le k \le K$) smallest element of $\boldsymbol{u} \in \mathbb{Z}^K$, implying $\langle \boldsymbol{u} \rangle_1 \le \langle \boldsymbol{u} \rangle_2 \le \cdots \le \langle \boldsymbol{u} \rangle_K$. Intuitively, $\boldsymbol{\langle u \rangle} = (\langle \boldsymbol{u} \rangle_1, \langle \boldsymbol{u} \rangle_2, \cdots, \langle \boldsymbol{u} \rangle_K)$ represents the non-decreasingly sorted version of $\boldsymbol{u}$.*

**Definition 2.** *For any $\boldsymbol{\alpha} \in \mathbb{Z}^K$ and $\boldsymbol{\beta} \in \mathbb{Z}^K$, if $\langle \boldsymbol{\alpha} \rangle_1 > \langle \boldsymbol{\beta} \rangle_1$ or $\exists k \in \{2, 3, \cdots, K\}$ such that $\langle \boldsymbol{\alpha} \rangle_k > \langle \boldsymbol{\beta} \rangle_k$ and $\langle \boldsymbol{\alpha} \rangle_i = \langle \boldsymbol{\beta} \rangle_i, \forall i \in \{1, \cdots, k-1\}$, then $\boldsymbol{\alpha}$ is* lexicographically greater *than $\boldsymbol{\beta}$, represented as $\boldsymbol{\alpha} \succ \boldsymbol{\beta}$. Similarly, if $\langle \boldsymbol{\alpha} \rangle_k = \langle \boldsymbol{\beta} \rangle_k, \forall k \in \{1, 2, \cdots, K\}$ or $\boldsymbol{\alpha} \succ \boldsymbol{\beta}$, then $\boldsymbol{\alpha}$ is* lexicographically no smaller *than $\boldsymbol{\beta}$, represented as $\boldsymbol{\alpha} \succeq \boldsymbol{\beta}$.*

**Definition 3.** *$\underset{\boldsymbol{x}}{lexmax}\ \boldsymbol{f}(\boldsymbol{x})$ represents the* lexicographical maximization *of the vector $\boldsymbol{f} \in \mathbb{R}^M$, which consists of $M$ objective functions of $\boldsymbol{x}$. To be particular, the optimal solution $\boldsymbol{x}^* \in \mathbb{R}^K$ achieves the optimal $\boldsymbol{f}^*$, in the sense that $\boldsymbol{f}^* = \boldsymbol{f}(\boldsymbol{x}^*) \succeq \boldsymbol{f}(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathbb{R}^K$.*

Let $x_{i,j}^{k,t}$ denote the number of data units of coflow $k$ that is transferred from server $i$ to $j$ at time slot $t$. We are now ready to formulate our utility optimal coflow scheduling as follows:

$$\underset{\boldsymbol{x}}{\text{lexmax}} \quad \boldsymbol{f} = (\mathcal{U}_1(\tau_1), \mathcal{U}_2(\tau_2), \cdots, \mathcal{U}_K(\tau_K)) \tag{4.1}$$

$$\text{s.t.} \quad \tau_k = \max\{t \in \mathcal{T} | x_{i,j}^{k,t} > 0, \forall i, j\}, \forall k \in \mathcal{K} \tag{4.2}$$

$$\sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{N}} x_{i,j}^{k,t} \le 1, \quad \forall i \in \mathcal{N}, \ \forall t \in \mathcal{T} \tag{4.3}$$

$$\sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{N}} x_{i,j}^{k,t} \le 1, \quad \forall j \in \mathcal{N}, \ \forall t \in \mathcal{T} \tag{4.4}$$

$$\sum_{t \in \mathcal{T}} x_{i,j}^{k,t} = D_{i,j}^k, \quad \forall i, j \in \mathcal{N}, \ \forall k \in \mathcal{K} \tag{4.5}$$

$$x_{i,j}^{k,t} \in \mathbb{Z}^+, \quad \forall i, j \in \mathcal{N}, \ \forall t \in \mathcal{T}, \ \forall k \in \mathcal{K} \tag{4.6}$$

where $\tau_k$ represents the completion time of coflow $k$, which is the time slot when the last unit of data that belongs to $k$ is transmitted, according to constraint (4.2). Constraint (4.3) indicates that the total number of data units transmitted through the outgoing link of server $i$ at time slot $t$ does not exceed the link capacity. Similarly, constraint (4.4) specifies that each incoming link transmits at most 1 unit of data per time slot. Constraint (4.5) implies the total amount of data units to be transmitted for each flow.

The objective is a vector $\boldsymbol{f} \in \mathbb{R}^K$ with $K$ elements, each standing for the utility of a particular coflow $k \in \mathcal{K}$. According to the previous definitions, the optimal $\boldsymbol{f}^*$ is lexicographically no smaller than any $\boldsymbol{f}$ obtained with a feasible schedule, which means that when sorting them in a non-descending order, if their $k$-th smallest element satisfies $\langle \boldsymbol{f}^* \rangle_{k'} = \langle \boldsymbol{f} \rangle_{k'}, \forall k' < k$ and $\langle \boldsymbol{f}^* \rangle_k \neq \langle \boldsymbol{f} \rangle_k$, then we have $\langle \boldsymbol{f}^* \rangle_k > \langle \boldsymbol{f} \rangle_k$. This implies that the first smallest element of $\boldsymbol{f}^*$, *i.e.*, the worst coflow utility, is the maximum among all $\boldsymbol{f}$. Then among all $\boldsymbol{f}$ with the same worst utility, the second worst utility in $\boldsymbol{f}^*$ is the maximum, and so on. In this way, solving this problem would result in an optimal schedule vector $\boldsymbol{x}^*$, with which all the coflow utilities are maximized.

## 4.3 Optimizing the Worst Utility among Concurrent Coflows

As interpreted in the previous section, Problem (4.1) is a vector optimization with multiple objectives. In this section, we consider the single-objective subproblem of optimizing the worst coflow utility as follows:

$$\begin{aligned} \max_{\boldsymbol{x}} \quad & \min_{k \in \mathcal{K}} \left( \mathcal{U}_k(\tau_k) \right) \qquad (4.7) \\ \text{s.t.} \quad & \text{Constraints } (4.2), (4.3), (4.4), (4.5) \text{ and } (4.6). \end{aligned}$$

which is the primary step for solving the original problem, to be elaborated in the next section.

This problem is an integer programming problem with a nonlinear constraint (4.2).

With an in-depth investigation of the problem structure, we transform Problem (4.7) into an equivalent linear programming (LP) problem that can be solved efficiently to obtain the optimal schedule vector $\boldsymbol{x}$. To be more specific, the features of *separable convex objective* and *totally unimodular linear constraints* are utilized in our transformation, which involves three major steps to be elaborated in the following subsections.

## 4.3.1 Totally Unimodular Linear Constraints

We first eliminate the nonlinear constraint (4.2) by expressing the utility of coflow $k \in \mathcal{K}$ directly with its scheduling variables $x_{i,j}^{k,t}, \forall i, j \in \mathcal{N}, \forall t \in \mathcal{T}$, rather than its completion time $\tau_k$.

As previously defined, $u_k^t$ is the utility value achieved by coflow $k$, if it completes at time slot $t$, *i.e.*, $u_k^t = \mathcal{U}_k(t)$. Since the utility function is non-increasing, we have $u_k^t \geq u_k^{t'}$ for $t < t'$. Let $\phi(x_{i,j}^{k,t})$ denote an indicator function, so that $\phi(x_{i,j}^{k,t}) = 1$ if $x_{i,j}^{k,t} > 0$, and $\phi(x_{i,j}^{k,t}) = 0$, otherwise. Hence, the utility of coflow $k$ can be represented as $\min_{i,j,t,x_{i,j}^{k,t} \neq 0} u_k^t \phi(x_{i,j}^{k,t})$, which correctly indicates that the coflow utility is the smallest utility value achieved at the maximum $t$ that satisfies $x_{i,j}^{k,t} \neq 0$, *i.e.*, the time slot when the last flow of $k$ completes. Therefore, Problem (4.7) can be transformed as follows with the nonlinear constraint (4.2) eliminated:

$$\begin{aligned} \max_{\boldsymbol{x}} \quad & \min_{k \in \mathcal{K}} \Big( \min_{i,j,t,x_{i,j}^{k,t} \neq 0} u_k^t \phi(x_{i,j}^{k,t}) \Big) \\ \text{s.t.} \quad & \text{Constraints } (4.3), (4.4), (4.5) \text{ and } (4.6). \end{aligned} \tag{4.8}$$

Now we investigate the coefficient matrix of linear constraints (4.3) (4.4) and (4.5). An $m$-by-$n$ matrix is *totally unimodular* [16], if it satisfies two conditions: 1) any of its elements belongs to $\{-1, 0, 1\}$; 2) any row subset $R \subset \{1, 2, \cdots, m\}$ can be divided into two disjoint sets, $R_1$ and $R_2$, such that $|\sum_{i \in R_1} a_{ij} - \sum_{i \in R_2} a_{ij}| \leq 1, \forall j \in \{1, 2, \cdots, n\}$.

**Lemma 1.** *The coefficients of constraints (4.3), (4.4) and (4.5) form a totally unimod-*

*ular matrix.*

*Proof.* Let $\mathbf{A}_{m\times n}$ denote the coefficient matrix of all the linear constraints (4.3), (4.4) and (4.5), where $m = 2NT + KN^2$, representing the total number of the constraints, and $n = KTN^2$, denoting the dimension of the variable $\boldsymbol{x}$.

It is obvious that any element of $\mathbf{A}_{m\times n}$ is either 0 or 1, satisfying the first condition. For any row subset $R \subset \{1, 2, \cdots, m\}$, to check the second condition, we consider the following cases:

Case I: If $Q_1 = \{1, 2, \cdots, NT\} \subseteq R$, then we can group $Q_1$ into $R_1$ and $R - Q_1$ into $R_2$. In this way, $\forall j \in \{1, 2, \cdots, n\}$, we have $\sum_{i\in R_1} a_{ij} = 1$, and $0 \leq \sum_{i\in R_2} a_{ij} \leq 2$, hence, $|\sum_{i\in R_1} a_{ij} - \sum_{i\in R_2} a_{ij}| \leq 1$, satisfying the second condition. In a similar vein, if $Q_2 = \{NT+1, NT+2, \cdots, 2NT\} \subseteq R$, then we can group $Q_2$ into $R_1$ and $R - Q_2$ into $R_2$; if $Q_3 = \{2NT+1, 2NT+2, \cdots, 2NT+KN^2\} \subseteq R$, then we can group $Q_3$ into $R_1$ and $R - Q_3$ into $R_2$.

Case II: If $Q_1 \nsubseteq R$ and $Q_2 \nsubseteq R$ and $Q_3 \nsubseteq R$ , we can divide $R$ with the following steps. Initialize $R' = R$ as the set of ungrouped rows. First, for any column $j$ from 1 to $n$, if $\sum_{i\in R} a_{ij} = 2$, $a_{i_1 j} = a_{i_2 j} = 1$, *i.e.*, there are two elements with value 1, we separate rows $i_1$ and $i_2$ into two sets and remove them from $R'$. To be specific, $i_1$ is grouped into $R_1$ if $\sum_{i\in(R_1+\{i_1\})} a_{il} \neq 3, \forall l \in \{1, 2, \cdots, j-1\}$; otherwise, $i_1$ is added to $R_2$. Second, for any column $j$ from 1 to $n$, if $\sum_{i\in R} a_{ij} = 3$, $a_{i_1 j} = a_{i_2 j} = a_{i_3 j} = 1$ and if $i_1 \in R'$ or $i_2 \in R'$ or $i_3 \in R'$, add the ungrouped row(s) so that $i_1, i_2$ and $i_3$ are not in the same subset, and remove them from $R'$. Finally, add the remaining rows in $R'$ to any of the subset. In this way, we can guarantee that for any column $j \in \{1, 2, \cdots, n\}$, if $\sum_{i\in R} a_{ij} = 2$, then $\sum_{i\in R_1} a_{ij} = \sum_{i\in R_2} a_{ij} = 1$; if $\sum_{i\in R} a_{ij} = 3$, then $1 \leq \sum_{i\in R_1} a_{ij} \leq 2$ and $1 \leq \sum_{i\in R_2} a_{ij} \leq 2$. Therefore, the second condition is satisfied.

In summary, we have shown that both conditions for total unimodularity are satisfied, thus the coefficient matrix $\mathbf{A}_{m\times n}$ is totally unimodular. □

## 4.3.2 Separable Convex Objective

In this subsection, we will show that the optimal solution for Problem (4.8) can be obtained by solving a problem with a separable convex objective function, which is represented as a summation of convex functions with respect to each single variable $x_{i,j}^{k,t}$.

We first show that the optimal solution of Problem (4.8) can be obtained by solving the following problem:

$$\begin{aligned}\underset{\boldsymbol{x}}{\text{lexmax}} \quad & \boldsymbol{g} = (u_1^1\phi(x_{1,1}^{1,1}), \cdots, u_k^t\phi(x_{i,j}^{k,t}), \cdots, u_K^T\phi(x_{N,N}^{K,T})) \\ \text{s.t.} \quad & \text{Constraints } (4.3), (4.4), (4.5) \text{ and } (4.6).\end{aligned}$$

where $\boldsymbol{g}$ is a vector with the dimension of $M = |\boldsymbol{g}| = KTN^2$. For any feasible $\boldsymbol{x}$, as the total number of data units to be transferred is fixed, it is intuitive that the total number of $x_{i,j}^{k,t}$ that equals to 0 is also fixed, denoted by $p$. Thus, any $\boldsymbol{g}$ achieved by a feasible $\boldsymbol{x}$ has at least $p$ elements of value 0. Consider the optimal $\boldsymbol{g}^*$ and any feasible $\boldsymbol{g}$, we have $\langle \boldsymbol{g}^* \rangle_k = \langle \boldsymbol{g} \rangle_k = 0, \forall k \leq p$, and $\langle \boldsymbol{g}^* \rangle_{p+1} \geq \langle \boldsymbol{g} \rangle_{p+1}$, where $\langle \boldsymbol{g}^* \rangle_{p+1} = \min_{i,j,t,x_{i,j}^{k,t} \neq 0} \ u_k^t$, representing the worst coflow utility. This indicates that $\boldsymbol{g}^*$ achieves the maximal worst coflow utility. Therefore, the optimal schedule $\boldsymbol{x}^*$ which gives $\boldsymbol{g}^*$ is also the optimal solution for Problem (4.8).

For ease of expression, we use $\mathcal{X}$ to represent the set of feasible $\boldsymbol{x}$ that satisfies constraints (4.3), (4.4), (4.5) and (4.6). We then transform the problem $\text{lexmax}_{\boldsymbol{x} \in \mathcal{X}} \ \boldsymbol{g}$ to an equivalent problem with a separable convex objective, based on the following lemmas.

**Lemma 2.** *$lexmax_{\boldsymbol{x} \in \mathcal{X}} \ \boldsymbol{g} \iff lexmin_{\boldsymbol{x} \in \mathcal{X}} \ -\boldsymbol{g}$.*

*Proof.* We first define the lexicographical order that is opposite to the direction in Definition 5 and 6. For a vector $\boldsymbol{\alpha} \in \mathbb{Z}^K$, $\langle \boldsymbol{\alpha} \rangle' = (\langle \boldsymbol{\alpha} \rangle_K, \langle \boldsymbol{\alpha} \rangle_{K-1}, \cdots, \langle \boldsymbol{\alpha} \rangle_1)$ represents the sorted version (in non-ascending order) of $\boldsymbol{\alpha}$. For any $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{Z}^K$, if the first non-zero element of $\langle \boldsymbol{\alpha} \rangle' - \langle \boldsymbol{\beta} \rangle'$ is negative, then $\boldsymbol{\alpha}$ is *lexicographically smaller* than $\boldsymbol{\beta}$, represented as $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$. If $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$ or $\langle \boldsymbol{\alpha} \rangle' = \langle \boldsymbol{\beta} \rangle'$, then $\boldsymbol{\alpha}$ is *lexicographically no greater* than $\boldsymbol{\beta}$, represented as $\boldsymbol{\alpha} \preceq \boldsymbol{\beta}$.

Note that $\boldsymbol{\alpha} \preceq \boldsymbol{\beta}$ does not imply $\boldsymbol{\beta} \succeq \boldsymbol{\alpha}$, or vice versa. The *lexicographical minimization* of $\boldsymbol{f}(\boldsymbol{x})$ over $\boldsymbol{x} \in \mathbb{Z}^K$, represented as $\operatorname*{lexmin}_{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x})$, is to find the optimal $\boldsymbol{x}^*$ that satisfies $\boldsymbol{f}^* = \boldsymbol{f}(\boldsymbol{x}^*) \preceq \boldsymbol{f}(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathbb{R}^K$.

Now we consider the optimal $\boldsymbol{x}^*$ to $\text{lexmax}_{\boldsymbol{x}\in\mathcal{X}}\ \boldsymbol{g}$. According to Definition 6, $\boldsymbol{g}(\boldsymbol{x}^*) \succeq \boldsymbol{g}(\boldsymbol{x})$, $\forall \boldsymbol{x} \in \mathcal{X}$, which implies that the first non-zero element of $\boldsymbol{\langle} \boldsymbol{g}(\boldsymbol{x}^*) \boldsymbol{\rangle} - \boldsymbol{\langle} \boldsymbol{g}(\boldsymbol{x}) \boldsymbol{\rangle}$ is positive, recalling that $\boldsymbol{\langle} \boldsymbol{g}(\boldsymbol{x}) \boldsymbol{\rangle} = (\langle \boldsymbol{g}(\boldsymbol{x}) \rangle_1, \langle \boldsymbol{g}(\boldsymbol{x}) \rangle_2, \cdots, \langle \boldsymbol{g}(\boldsymbol{x}) \rangle_K)$. Since $\langle \boldsymbol{g}(\boldsymbol{x}) \rangle_1$ is the minimum element in $\boldsymbol{g}(\boldsymbol{x})$, it is obvious that $-\langle \boldsymbol{g}(\boldsymbol{x}) \rangle_1$ is the maximum element in $-\boldsymbol{g}(\boldsymbol{x})$, *i.e.*, $\langle -\boldsymbol{g}(\boldsymbol{x}) \rangle_K = -\langle \boldsymbol{g}(\boldsymbol{x}) \rangle_1$. The same applies to all the elements in $\boldsymbol{g}(\boldsymbol{x})$, which gives $\boldsymbol{\langle} -\boldsymbol{g}\ (\boldsymbol{x}) \boldsymbol{\rangle}' = (\langle -\boldsymbol{g}(\boldsymbol{x}) \rangle_K, \langle -\boldsymbol{g}(\boldsymbol{x}) \rangle_{K-1}, \cdots, \langle -\boldsymbol{g}(\boldsymbol{x}) \rangle_1) = (-\langle \boldsymbol{g}(\boldsymbol{x}) \rangle_1, -\langle \boldsymbol{g}(\boldsymbol{x}) \rangle_2, \cdots, -\langle \boldsymbol{g}(\boldsymbol{x}) \rangle_K) = -\boldsymbol{\langle} \boldsymbol{g}(\boldsymbol{x}) \boldsymbol{\rangle}$. Hence, we have $\boldsymbol{\langle} -\boldsymbol{g}\ (\boldsymbol{x}^*) \boldsymbol{\rangle}' - \boldsymbol{\langle} -\boldsymbol{g}\ (\boldsymbol{x}) \boldsymbol{\rangle}' = -(\ \boldsymbol{\langle} \boldsymbol{g}(\boldsymbol{x}^*) \boldsymbol{\rangle} - \boldsymbol{\langle} \boldsymbol{g}(\boldsymbol{x}) \boldsymbol{\rangle}\ )$, of which the first non-zero element is negative, indicating that $-\boldsymbol{g}(\boldsymbol{x}^*) \preceq -\boldsymbol{g}(\boldsymbol{x})$, $\forall \boldsymbol{x} \in \mathcal{X}$.

As such, we have proved that

$$\boldsymbol{g}(\boldsymbol{x}^*) \succeq \boldsymbol{g}(\boldsymbol{x}) \iff -\boldsymbol{g}(\boldsymbol{x}^*) \preceq -\boldsymbol{g}(\boldsymbol{x}),\ \forall \boldsymbol{x} \in \mathcal{X},$$

and thus the lemma holds obviously with the definitions of $\text{lexmax}_{\boldsymbol{x}\in\mathcal{X}}\ \boldsymbol{g}$ and $\text{lexmin}_{\boldsymbol{x}\in\mathcal{X}}\ -\boldsymbol{g}$.

□

Let $\varphi(\boldsymbol{g})$ define a function of $\boldsymbol{g}$:

$$\varphi(\boldsymbol{g}) = \sum_{m=1}^{|\boldsymbol{g}|} |\boldsymbol{g}|^{g_m} = \sum_{m=1}^{M} M^{g_m}$$

where $g_m$ is the $m$-th element of $\boldsymbol{g}$.

**Lemma 3.** *$\varphi(\cdot)$ preserves the order of* lexicographically no greater *($\preceq$),* i.e.*, $-\boldsymbol{g}(\boldsymbol{x}^*) \preceq -\boldsymbol{g}(\boldsymbol{x}) \iff \varphi(-\boldsymbol{g}(\boldsymbol{x}^*)) \leq \varphi(-\boldsymbol{g}(\boldsymbol{x}))$.*

*Proof.* We first consider $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{Z}^K$ that satisfies $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$. If we use the integer $\tilde{k}(1 \leq \tilde{k} \leq K)$ to represent the first non-zero element of $\boldsymbol{\langle\alpha\rangle}' - \boldsymbol{\langle\beta\rangle}'$, we have $\langle \boldsymbol{\alpha} \rangle_{\tilde{k}} < \langle \boldsymbol{\beta} \rangle_{\tilde{k}}$.

If $\tilde{k} = K$, assume $\langle \boldsymbol{\alpha} \rangle_K = n$, then $\langle \boldsymbol{\beta} \rangle_K \geq n + 1$.

$$\begin{aligned} \varphi(\boldsymbol{\alpha}) &= \sum_{k=1}^{K} K^{\langle \boldsymbol{\alpha} \rangle_k} = K^{\langle \boldsymbol{\alpha} \rangle_K} + \sum_{k=1}^{K-1} K^{\langle \boldsymbol{\alpha} \rangle_k} \\ &\leq K^{\langle \boldsymbol{\alpha} \rangle_K} + (K-1)K^{\langle \boldsymbol{\alpha} \rangle_K} = K \cdot K^{\langle \boldsymbol{\alpha} \rangle_K} = K^{n+1}, \end{aligned}$$

where the inequality holds because $\langle \boldsymbol{\alpha} \rangle_K \geq \langle \boldsymbol{\alpha} \rangle_k, \forall 1 \leq k \leq K-1$.

$$\begin{aligned} \varphi(\boldsymbol{\beta}) &= \sum_{k=1}^{K} K^{\langle \boldsymbol{\beta} \rangle_k} = K^{\langle \boldsymbol{\beta} \rangle_K} + \sum_{k=1}^{K-1} K^{\langle \boldsymbol{\beta} \rangle_k} \\ &> K^{\langle \boldsymbol{\beta} \rangle_K} + (K-1) \cdot 0 \geq K^{n+1}. \end{aligned}$$

Hence, we have $\varphi(\boldsymbol{\alpha}) < \varphi(\boldsymbol{\beta})$ for the case of $\tilde{k} = K$.

If $\tilde{k} < K$, we have $\langle \boldsymbol{\alpha} \rangle_k = \langle \boldsymbol{\beta} \rangle_k, \forall \tilde{k} \leq k \leq K$ and $\langle \boldsymbol{\alpha} \rangle_{\tilde{k}} < \langle \boldsymbol{\beta} \rangle_{\tilde{k}}$. Assume $\langle \boldsymbol{\alpha} \rangle_{\tilde{k}} = m$, then $\langle \boldsymbol{\beta} \rangle_{\tilde{k}} \geq m + 1$.

$$\begin{aligned} \varphi(\boldsymbol{\alpha}) &= \sum_{k=1}^{K} K^{\langle \boldsymbol{\alpha} \rangle_k} = \sum_{k=\tilde{k}+1}^{K} K^{\langle \boldsymbol{\alpha} \rangle_k} + K^{\langle \boldsymbol{\alpha} \rangle_{\tilde{k}}} + \sum_{k=1}^{\tilde{k}-1} K^{\langle \boldsymbol{\alpha} \rangle_k} \\ &\leq \sum_{k=\tilde{k}+1}^{K} K^{\langle \boldsymbol{\alpha} \rangle_k} + K^{\langle \boldsymbol{\alpha} \rangle_{\tilde{k}}} + (\tilde{k}-1)K^{\langle \boldsymbol{\alpha} \rangle_{\tilde{k}}} \\ &= \sum_{k=\tilde{k}+1}^{K} K^{\langle \boldsymbol{\alpha} \rangle_k} + \tilde{k} \cdot K^{\langle \boldsymbol{\alpha} \rangle_{\tilde{k}}} = \sum_{k=\tilde{k}+1}^{K} K^{\langle \boldsymbol{\alpha} \rangle_k} + \tilde{k}K^m, \end{aligned}$$

where the inequality holds as $\langle\boldsymbol{\alpha}\rangle_{\tilde{k}} \geq \langle\boldsymbol{\alpha}\rangle_k, \forall 1 \leq k \leq \tilde{k}-1$.

$$\begin{aligned}
\varphi(\boldsymbol{\beta}) &= \sum_{k=1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k} = \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k} + K^{\langle\boldsymbol{\beta}\rangle_{\tilde{k}}} + \sum_{k=1}^{\tilde{k}-1} K^{\langle\boldsymbol{\beta}\rangle_k} \\
&> \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k} + K^{\langle\boldsymbol{\beta}\rangle_{\tilde{k}}} + (\tilde{k}-1)\cdot 0 \\
&\geq \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k} + K^{m+1} > \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k} + \tilde{k}K^{m}.
\end{aligned}$$

Given that $\sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\alpha}\rangle_k} = \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k}$, we have $\varphi(\boldsymbol{\alpha}) < \varphi(\boldsymbol{\beta})$ for the case of $\tilde{k} < K$.

If $\boldsymbol{\alpha} = \boldsymbol{\beta}$, which means that $\langle\boldsymbol{\alpha}\rangle_k = \langle\boldsymbol{\beta}\rangle_k, \forall 1 \leq k \leq K$, it is trivially true that $\varphi(\boldsymbol{\alpha}) = \sum_{k=1}^{K} K^{\langle\boldsymbol{\alpha}\rangle_k} = \sum_{k=1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k} = \varphi(\boldsymbol{\beta})$. Thus, we have proved $\boldsymbol{\alpha} \preceq \boldsymbol{\beta} \Longrightarrow \varphi(\boldsymbol{\alpha}) \leq \varphi(\boldsymbol{\beta})$.

We further prove $\varphi(\boldsymbol{\alpha}) \leq \varphi(\boldsymbol{\beta}) \Longrightarrow \boldsymbol{\alpha} \preceq \boldsymbol{\beta}$ by proving its contrapositive: $\neg(\boldsymbol{\alpha} \preceq \boldsymbol{\beta}) \Longrightarrow \varphi(\boldsymbol{\alpha}) > \varphi(\boldsymbol{\beta})$. $\neg(\boldsymbol{\alpha} \preceq \boldsymbol{\beta})$ implies $\boldsymbol{\alpha} \neq \boldsymbol{\beta}$ and the first non-zero element of $\langle\boldsymbol{\alpha}\rangle' - \langle\boldsymbol{\beta}\rangle'$ is positive, which further indicates the first non-zero element of $\langle\boldsymbol{\beta}\rangle' - \langle\boldsymbol{\alpha}\rangle'$ is negative, *i.e.*, $\boldsymbol{\beta} \prec \boldsymbol{\alpha}$. Thus, the contrapositive is equivalent to $\boldsymbol{\beta} \prec \boldsymbol{\alpha} \Longrightarrow \varphi(\boldsymbol{\beta}) < \varphi(\boldsymbol{\alpha})$, which has already been proved previously using the exchanged notations of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

With $\boldsymbol{\alpha} \preceq \boldsymbol{\beta} \iff \varphi(\boldsymbol{\alpha}) \leq \varphi(\boldsymbol{\beta})$ holding for any $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ of the same dimension, we are done with the proof by letting $\boldsymbol{\alpha} = -\boldsymbol{g}(\boldsymbol{x}^*)$ and $\boldsymbol{\beta} = -\boldsymbol{g}(\boldsymbol{x})$. □

Based on Lemma 2 and Lemma 4, we have

$$\operatorname*{lexmax}_{\boldsymbol{x}\in\mathcal{X}} \ \boldsymbol{g} \iff \min_{\boldsymbol{x}\in\mathcal{X}} \ \varphi(-\boldsymbol{g}) = \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}} M^{-u_k^t \phi(x_{i,j}^{k,t})}$$

where the objective function $\varphi(-\boldsymbol{g})$ is a summation of the term $M^{-u_k^t \phi(x_{i,j}^{k,t})}$, which is a convex function of the single variable $x_{i,j}^{k,t}$.

Therefore, solving Problem (4.8) is equivalent to solving the following problem with a separable convex objective:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}} M^{-u_k^t \phi(x_{i,j}^{k,t})} \qquad (4.9) \\ \text{s.t.} \quad & \text{Constraints } (4.3), (4.4), (4.5) \text{ and } (4.6). \end{aligned}$$

### 4.3.3 Structure-Inspired Equivalent LP Transformation

Exploiting the problem structure of totally unimodular constraints and separable convex objective, we can use the $\lambda$-representation technique [64] to transform the Problem (4.9) to a linear programming problem that has the same optimal solution.

For each single integer variable $x_{i,j}^{k,t} \in \{0,1\}$, the convex function $h_{i,j}^{k,t}(x_{i,j}^{k,t}) = M^{-u_k^t \phi(x_{i,j}^{k,t})}$ can be linearized with the $\lambda$-representation as follows:

$$h_{i,j}^{k,t}(x_{i,j}^{k,t}) = \sum_{s\in\{0,1\}} M^{-u_k^t s}\lambda_{i,j}^{k,t,s} = \lambda_{i,j}^{k,t,0} + M^{-u_k^t}\lambda_{i,j}^{k,t,1}$$

which removes the variable $x_{i,j}^{k,t}$ by sampling at each of its possible value $s \in \{0,1\}$, weighted by the newly introduced variables $\lambda_{i,j}^{k,t,s} \in \mathbb{R}^+, \forall s \in \{0,1\}$ that satisfy

$$x_{i,j}^{k,t} = \sum_{s\in\{0,1\}} s\lambda_{i,j}^{k,t,s} = \lambda_{i,j}^{k,t,1}$$

$$\sum_{s\in\{0,1\}} \lambda_{i,j}^{k,t,s} = \lambda_{i,j}^{k,t,0} + \lambda_{i,j}^{k,t,1} = 1$$

Further, with linear relaxation on the integer constraints (4.6), we obtain the following linear programming problem:

$$
\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{\lambda}} \quad & \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}}(\lambda_{i,j}^{k,t,0} + M^{-u_k^t}\lambda_{i,j}^{k,t,1}) \qquad (4.10)\\
\text{s.t.} \quad & x_{i,j}^{k,t} = \lambda_{i,j}^{k,t,1}, \ \ \forall i,j\in\mathcal{N}, k\in\mathcal{K}, t\in\mathcal{T}\\
& \lambda_{i,j}^{k,t,0} + \lambda_{i,j}^{k,t,1} = 1, \ \ \forall i,j\in\mathcal{N}, k\in\mathcal{K}, t\in\mathcal{T}\\
& \lambda_{i,j}^{k,t,0}, \lambda_{i,j}^{k,t,1}, x_{i,j}^{k,t} \in \mathbb{R}^+, \ \forall i,j\in\mathcal{N}, k\in\mathcal{K}, t\in\mathcal{T}\\
& \text{Constraints } (4.3), (4.4) \text{ and } (4.5).
\end{aligned}
$$

**Theorem 2.** *An optimal solution to Problem (4.10) is an optimal solution to Problem (4.7).*

*Proof.* The property of total unimodularity ensures that an optimal solution to the relaxed LP problem (4.10) has integer values of $x_{i,j}^{k,t}$, which is an optimal solution to Problem (4.9), and thus an optimal solution to Problem (4.8) as demonstrated in the previous subsection. Moreover, Problem (4.7) and (4.8) are equivalent forms, completing the proof. □

Therefore, an optimal schedule that maximizes the worst utility among all the coflows can be obtained by solving Problem (4.10) with efficient LP solvers, such as MOSEK [17].

## 4.4 Iteratively Optimizing Worst Utilities to Achieve Max-Min Fairness

With the subproblem of maximizing the worst utility efficiently solved as an LP problem (4.10), we continue to solve our original multi-objective problem (4.1) by maximizing the next worst utility repeatedly.

After solving the subproblem, it is known that the optimal worst utility is achieved by coflow $k^*$ at time slot $t^*$, when its slowest flow from server $i^*$ to $j^*$ completes. We then fix the computed schedule of all the $i^*$-to-$j^*$ flows at time $t^*$, which means that the corresponding schedule variables $x_{i^*,j^*}^{k,t^*}, \forall k \in \mathcal{K}$ are removed from the variable set $\boldsymbol{x}$ for the next round. Also, since coflow $k^*$ completes at time $t^*$, it is intuitive that all the scheduling variables associated with it after time $t^*$ should be fixed as zero and removed: $x_{i,j}^{k^*,t} = 0, \forall i, j \in \mathcal{N}, t \in \{t^* + 1, t^* + 2, \cdots, T\}$.

As we have fixed a part of the schedule, link capacities and remaining flow sizes should be updated in problem constraints in the next round. For example, if $x_{i^*,j^*}^{1,1} = 1$, which means that at time slot 1, the $i^*$-to-$j^*$ flow that belongs to coflow 1 would be scheduled, then for the problem in the next round, $x_{i^*,j^*}^{1,1}$ is no longer the variable. The link capacity constraints should be updated as $\sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{N},(j,k)\neq(j^*,1)} x_{i^*,j}^{k,1} \leq 0$ and $\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{N},(i,k)\neq(i^*,1)} x_{i,j^*}^{k,1} \leq 0$, which means that there is no capacity at the outgoing link of server $i^*$ and the incoming link of server $j^*$ at time slot 1 to schedule other flows; the flow size capacity constraint should be updated as $\sum_{t\in\{2,3,\cdots,T\}} x_{i^*,j^*}^{1,t} = D_{i^*,j^*}^{1} - 1$.

Moreover, the utility of coflow $k^*$ is obtained as $u_{k^*}^{t^*}$, yet the schedules of all its flows except the slowest have not been fixed, which would be the variables $(x_{i,j}^{k^*,t}, \forall (i,j) \neq (i^*, j^*), \forall t \in \{1, 2, \cdots, t^*\})$ of the problem in the next round. We set the associated utilities of these variables as $u_{k^*}^{t^*} x_{i,j}^{k^*,t}$. The rationale is that no matter how fast other flows of $k^*$ complete, the utility is determined by the slowest flow that finishes at $t^*$. This ensures that the utility optimized in the next round is achieved by another coflow, rather than a flow of coflow $k^*$ (other than its slowest). This is better illustrated with the example in Fig. 4.6.

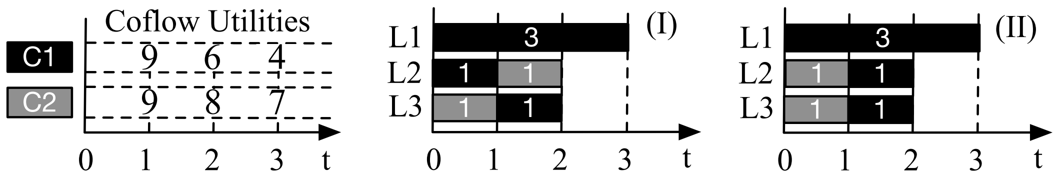


Figure 4.6: Utilities and two possible schedules of coflows $C1$ and $C2$.

It is obvious that coflow $C1$ achieves the worst utility of 4. In the second round, if we do not change the associated utility for the flows of $C1$ except the slowest, schedule

**Algorithm 3:** Utitity Optimal Schedule among Coflows with Max-Min Fairness.

```
Input:
    Coflow traffic matrix $\mathbf{D}^k = \left(D_{i,j}^k\right)_{i,j=1}^N$; Time slots $\mathcal{T} = \{1, 2, \cdots, T\}$; Coflow utility
    $u_k^t,\ \forall k \in \mathcal{K}, \forall t \in \mathcal{T}$;
Output:
    Flow schedule at each time slot: $x_{i,j}^{k,t}, \forall k \in \mathcal{K}, \forall t \in \mathcal{T}$;
1: Initialize $\mathcal{K}' = \mathcal{K}$;
2: while $\mathcal{K}' \neq \varnothing$ do
3:     Solve the LP Problem (4.10) to obtain the solution $\boldsymbol{x}$;
4:     Obtain $x_{k^*,i^*}^{j^*,t^*} = \underset{x_{i,j}^{k,t} \in \boldsymbol{x}}{\text{argmin}}\ u_k^t x_{i,j}^{k,t}$;
5:     Fix $x_{i,j}^{k^*,t}, \forall i, j \in \mathcal{N}, t \in \{t^*+1, t^*+2, \cdots, T\}$ and $x_{i^*,j^*}^{k,t^*},\ \forall k \in \mathcal{K}$; remove them
       from variable set $\boldsymbol{x}$;
6:     Update corresponding link capacities in Constraints (4.3), (4.4) and flow sizes in
       Constraints (4.5);
7:     Set $u_{k^*}^t = u_{k^*}^{t^*},\ \forall t \in \{1, 2, \cdots, t^*\}$;
8:     Remove $k^*$ from $\mathcal{K}'$;
9: end while
```

(I) would result in $(4, 6, 6, 8, 9, 9, 9)$ while schedule (II) gives $(4, 6, 6, 6, 9, 9, 9)$. Thus, the optimization would result in schedule (I) to achieve the optimal next worst utility as 8 for another flow of $C1$, which does not match our original objective to optimize the second worst utility for coflow $C2$. In contrast, if we set the associated utilities of all $C1$'s flows as 4, then schedule (II) would give $(4, 4, 4, 4, 4, 9, 9)$, better than $(4, 4, 4, 4, 4, 8, 9)$ given by schedule (I). In this way, the optimization can correctly choose schedule (II) to achieve the optimal utility of $C2$.

As a result, the subproblem in the next round is solved over a decreased set of variables with updated constraints and objectives, so that the next worst coflow utility would be optimized, without impacting the worst coflow utility in this round. Such procedure is repeatedly executed until the last worst utility of coflow has been optimized, and the max-min fairness has been achieved, as summarized in Algorithm 3.

## 4.5 Real-Wrold Implementation and Performance Evaluation

Having proved the theoretical optimality of our solution, we proceed to implement a practical coflow scheduler in the real world and demonstrate its effectiveness in optimizing coflow utilities.

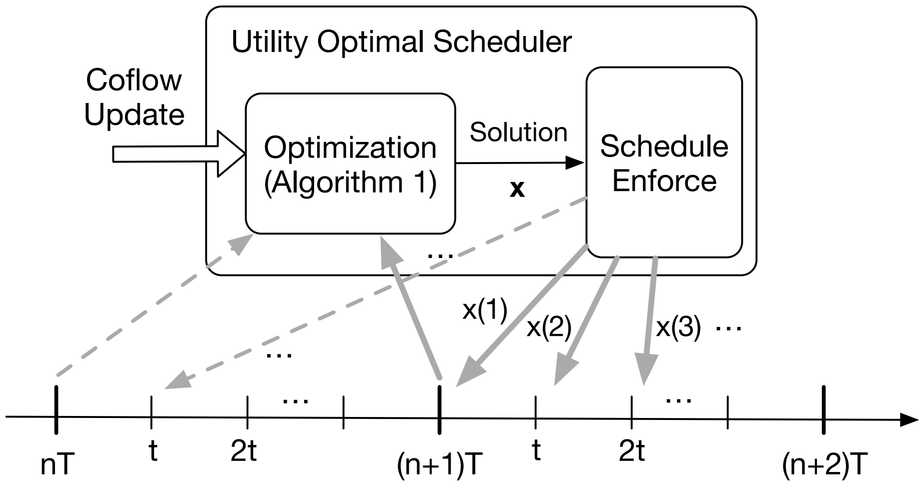


Figure 4.7: An illustration of the signaling cycle of utility optimal scheduler.

**Design and Implementation.** Varys [7] is an open-source framework that provides a simple API to data parallel jobs for coflow submission, and coordinates competing coflows through bandwidth allocation at the application layer. Our scheduler is implemented in the context of Varys to leverage the global view of the network and coflow information provided by the framework. As a natural fit for the time-slotted theoretical model, we design our scheduler to calculate and enforce scheduling decisions based on two-level time-triggered events, as shown in Fig. 4.7. The large time interval $T$ consists of a certain number of small intervals $t$ ($T = mt, m \in \mathbb{Z}^+$), responsible for decision making and enforcement, respectively. The small interval $t$ is the duration of time required to transfer a data block of a specific size at the full line rate. This corresponds to the length of time slot in our theoretical model. At each large interval, the scheduler invokes Algorithm 3 to calculate the scheduling decisions for all the competing coflows, based on the updated information on flow sizes and coflow utilities. With the input readily available, the linear programming problem is formulated and solved using the simplex method implemented in the `Breeze` optimization library [65]. The calculated decisions will then be enforced by application-layer bandwidth allocation slot by slot, updating the rate limit to the `ThrottledInputStream` (java I/O object) for each flow at each small interval. These parameters are tuneable, with the tradeoff that a smaller $T$ would be more responsive to flow dynamics, yet incurring more computation overhead to the scheduler.

**Meaningful Utility Functions.** The utility function of each coflow can be flexibly specified by a utility type and several parameters, with specific practical implications. In our experiment, we consider four types: 1) *constant* utility for background coflows, 2) *linear* utility for time-sensitive coflows, 3) *sigmoid* utility for time-sensitive coflows, and 4) *all-or-nothing* utility for time-critical coflows. Each type of utility function is defined by a few parameters, which can be set according to the desired completion time and the priority. For example, the sigmoid function has the following form:

$$\mathcal{U}(t) = \frac{p_1}{1 + e^{p_2 * (t - p_3)}}$$

where $p_1$ specifies the priority of the coflow, $p_2$ is a decay factor of the utility function, quantifying the degree of sensitivity to the coflow completion time (CCT), and $p_3$ denotes the desired completion time, which is the time to complete the coflow without any network contention.

For each coflow, the starting time of its utility function is the time when it is registered in the scheduler. As the utility decreases along the timeline, it accurately characterizes critical information, such as the arrival time and the sensitivity degree, for the scheduler to make the optimal decisions. As an example, for two identical coflows arriving at the same time, which have sigmoid utility functions with different decay factors, the one that decreases faster would be considered with a higher priority in the scheduler; for two identical coflows with the same utility function, which arrive at different times, the one that arrives earlier would be scheduled ahead of the other one, which naturally avoids starvation.

**Experiment Setup.** We deploy our utility optimal scheduler on a cluster of 6 Virtual Machine instances on Google's Cloud Compute Engine, with a total of 24 CPU cores and 156 GB memory. To allow an in-depth evaluation and analysis of our scheduler, we examine the scheduling process of 6 coflows illustrated in Table 4.1, in comparison with the Varys Shortest-Effective-Bottleneck-First (SEBF) scheduler [7], which is a state-of-the-art coflow scheduler to minimize CCTs. For simplicity, we choose 1 GB as the size of the block to be sent with full bandwidth at each time slot. Based on the measured bandwidth, the length of the slot, *i.e.*, the small time interval, is 3 seconds, and each large interval has 10 slots.

Table 4.1: 6 coflows with 4 types of utility functions for case study.

| Coflow Id | Flow Id | Src | Dst | Volume (GB) | Utility Function ($t$: slots) |
|---|---|---|---|---|---|
| C1 | 1 | S1 | S0 | 2 | $\mathcal{U}(t) = 1.3 - 0.1 * t$ |
| | 2 | S1 | S5 | 1 | |
| C2 | 1 | S1 | S4 | 1 | $\mathcal{U}(t) = 1.02 - 0.01 * t$ |
| | 2 | S1 | S5 | 2 | |
| C3 | 1 | S3 | S0 | 1 | $\mathcal{U}(t) = 1$, if $t < 4$; $\mathcal{U}(t) = 0$, otherwise. |
| | 2 | S3 | S1 | 1 | |
| | 3 | S3 | S2 | 2 | |
| C4 | 1 | S4 | S2 | 2 | $\mathcal{U}(t) = \frac{2}{1+e^{0.1*(t-2)}}$ |
| C5 | 1 | S5 | S1 | 2 | $\mathcal{U}(t) = \frac{2}{1+e^{1.1*(t-4)}}$ |
| | 2 | S5 | S2 | 2 | |
| C6 | 1 | S1 | S3 | 1 | $\mathcal{U}(t) = 1$ |
| | 2 | S1 | S4 | 1 | |

To be more representative, the utility functions of the coflows in our case study cover all the four types. Particularly, coflow C6 is a CCT insensitive background coflow, while C3 is time critical with a deadline. The other four are time sensitive coflows, with linear or sigmoid utility functions. C1 is more sensitive than C2, and the same applies to C5 versus C4. The parameters are set with the consideration of target completion times and sensitivity degrees. For example, C1 has a total of 3 blocks to be transferred. It would achieve a utility of 1 if it completes exactly with 3 time slots, which is the target completion time. Similarly, with 2 blocks of data, C4 would obtain a utility of 1 if it finishes at the end of the second time slot. Failing to complete by the target time would result in a decrease of utility at different rates, in the linear or sigmoid form, as shown in Table 4.1.

The comparison of coflow utilities obtained by our utility optimal scheduler and the Varys SEBF scheduler is presented in Fig. 4.8, which is the average of 5 runs. It is clearly shown that our scheduler improves the worst utility among all the coflows by nearly 70%, from 0.4887 achieved by C5 to 0.8377 of C4. Moreover, with our scheduler, 2 coflows have achieved a significant increase of their utilities, 2 coflows keep nearly the same utilities, while only 1 coflow experienced a slight utility decrease.

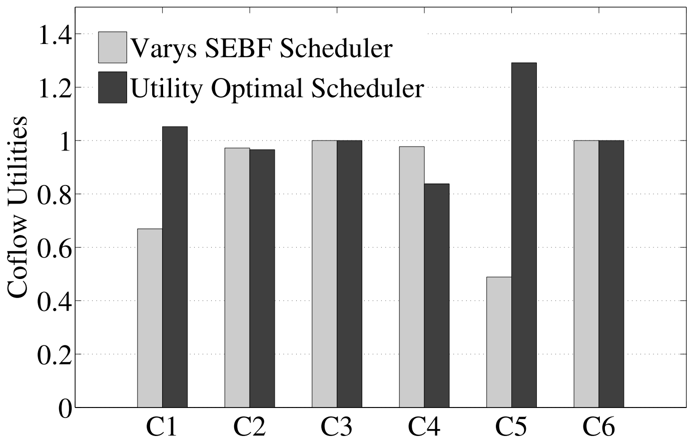


Figure 4.8: Comparison of utilities achived by coflows with two schedulers.

To clearly illustrate the underlying reason for such an improvement, we next provide a detailed analysis on how our scheduler provides the differential treatment to these coflows appropriately, according to their degrees of sensitivity to CCTs. At server S1, the outgoing link is bottlenecked by flows of C1, C2 and C6. According to our utility optimal schedule presented in Fig. 4.9, C1 with a higher degree of CCT sensitivity is scheduled first at S1, followed by C2 whose utility decreases much more slowly. C6 always achieves the constant utility so that it is scheduled at last when the link of S1 is idle. At server S2, C3 and C5 occupies the incoming link during the first 4 time slots in an interleaved manner, so that they both complete by their target time. As a result, C4 is delayed to complete at the 6-th slot. Such a schedule is utility optimal, since C4 is much more tolerant to the delay, with a small decay factor in its sigmoid utility function. If C4 is scheduled before C3 or C5, it results in either a zero utility for the time critical C3, or a much smaller utility value of C5, whose utility decreases sharply with a large decay factor.

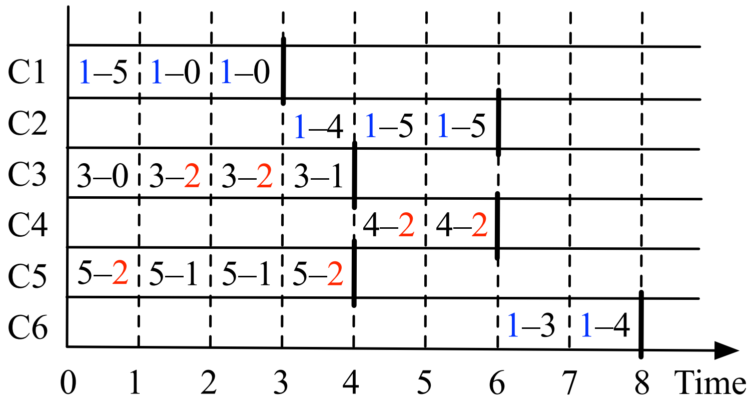


Figure 4.9: Utility optimal schedule. The 1–5 block of C1 represents a block of data sent from S1 to S5 in the first time slot. The bottleneck links are highlighted with blue and red colors, respectively.

In comparison to our utility optimal schedule, the SEBF scheduler in Varys for minimizing CCTs treats all the coflows equally, as if they all have the same linear utility function. Without the proper differentiation among the diverse degrees of CCT sensitivity, it schedules the background coflow C6 ahead of the time sensitive C1, which results in worse utilities, despite the improved CCT. In a similar vein, the less CCT sensitive C4 completes faster than C5 with the Varys scheduler, which is not utility optimal either.

Although we consider an offline case for optimality analysis, our scheduler is readily adaptive to the online scenario, where Algorithm 3 can be invoked to recalculate the optimal schedule based on the latest information as flow arrives and leaves.

## 4.6 Summary

In this chapter, we use utility functions to model different levels of sensitivity to the completion times of different coflows, in the context of data parallel jobs. With coflows competing for the network bandwidth in a shared cluster, we have designed and implemented a new utility optimal coflow scheduler to better satisfy their requirements with max-min fairness. To achieve this objective, we first formulated a lexicographical maximization problem to optimize all the coflow utilities, which is challenging due to the inherent complexity of both multi-objective and discrete optimizations. Starting from the single-objective subproblem and based on an in-depth investigation of the problem structure, we performed a series of transformations to finally obtain an equivalent linear programming (LP) problem, which can be efficiently solved in practice with a standard LP solver. With our algorithm repeatedly solving updated LP subproblems, we can optimize all the coflow utilities, with max-min fairness achieved. Last but not the least, we have implemented our utility optimal scheduler and demonstrated convincing evidence on the effectiveness of our new algorithm using real-world experiments.

# Chapter 5

# Optimal Job Scheduling across Datacenters

Large volumes of data are increasingly generated across multiple datacenters around the world, which is typical for global services deployed by Microsoft and Google generating user activity and system monitoring logs. In this chapter, we extend our consideration to the wide area with geographically distributed datacenters and address the challenging issue on optimizing data analytic jobs deployed in such an inter-datacenter network.

To process geo-distributed data, naively gathering all the data within a single datacenter is highly inefficient, as it incurs huge amounts of data to be transferred through narrow inter-datacenter links. As such, designing the best possible task assignment strategy to assign tasks to datacenters is critical, since different strategies lead to different flow patterns across datacenters, and ultimately, different job completion times.

Existing works in the literature only considered a single data analytic job when designing optimal task assignment strategies. Differently, we investigate the problem of assigning tasks belonging to multiple jobs across datacenters. As concurrent jobs are inherently competing for the limited amount of datacenter resources with each other, it is important to maintain fairness, which cannot be achieved if tasks from one job are assigned without considering the other jobs. We design a new task assignment strategy to achieve max-min fairness across multiple jobs with respect to their performance. We rigorously formulate the scheduling problem, address the challenges brought by the integer variables and transform the problem to a linear programming problem with proved equivalence. Our strategy is implemented and evaluated with real-world experiments, demonstrated to be effective in minimizing the job completion times across all concurrent jobs while maintaining max-min fairness.

## 5.1 Background and Motivation

In this section, we first present an overview of the execution of a data analytic job whose input data is stored across geographically distributed datacenters. We then consider the sharing of resources in these datacenters across multiple concurrent jobs, and provide a motivating example to illustrate the need for fair job scheduling.

### 5.1.1 Data Analytic Jobs in the Wide Area

It is typical for a data analytic job to contain tens or hundreds of tasks, supported by a data parallel framework, such as MapReduce and Spark. These tasks are parallel to or dependent upon each other, and network flows are generated between dependent tasks, since tasks in the subsequent stage need to fetch intermediate data from tasks in the current stage. For example, in a MapReduce job, a set of `map` tasks are first launched to read input data partitions and generate intermediate results; then `reduce` tasks would fetch such intermediate data from `map` tasks for further processing, which involves transferring data over the network.

In the case of running tasks in a data analytic job across multiple datacenters, data may be transferred over inter-datacenter links, which may become bottlenecks due to their limited bandwidth availability. Our design objective in this chapter is to compute the best way to assign tasks belonging to *multiple* jobs to geo-distributed datacenters, so that all jobs can achieve their best possible performance with respect to their completion times, without harming the performance of others. This implies that *max-min fairness* needs to be achieved across jobs sharing the datacenters, in terms of their job completion times.

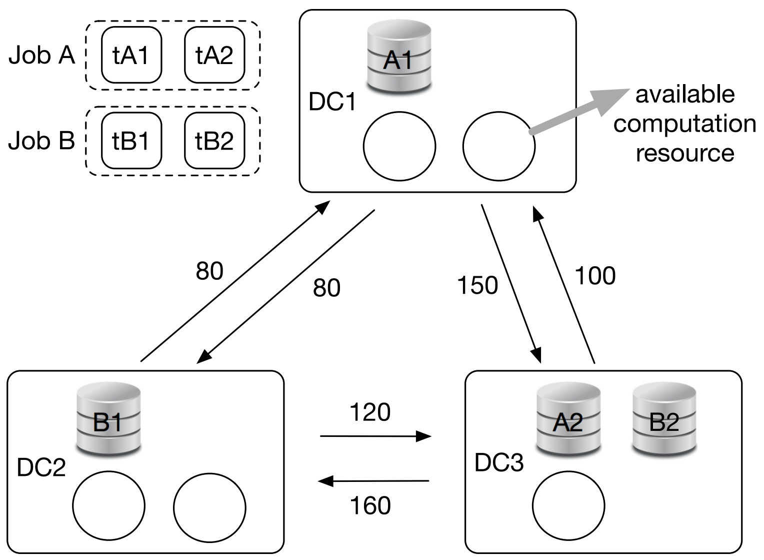


Figure 5.1: An example of scheduling multiple jobs fairly across geo-distributed datacenters.

For a better intuition of our problem, we use Fig. 5.1 to show an example with two data analytic jobs sharing three geo-distributed datacenters. For job `A`, both of its tasks, `tA1` and `tA2`, require 100 MB of data from input dataset `A1` stored in `DC1`, and 200 MB of data from `A2` located at `DC3`. For job `B`, the amounts of data to be read by task `tB1` from dataset `B1` in `DC2` and `B2` in `DC3` are both 200 MB; while task `tB2` needs to read 200 MB of data from `B1` and 300 MB from `B2`. These tasks are to be assigned to available computing slots in the three datacenters, each with two slots, two slots, and one slot, respectively. The amounts of available bandwidth of inter-datacenter links are illustrated in the figure, with the unit of MB/s.

For each job, a different assignment of its tasks will lead to flows of different sizes traversing different links, thus resulting in different job completion times. Moreover, both jobs must share and compete for the same pool of computing resources across these datacenters. For example, since `DC3` only has one available computing slot, if we assign a task from one job, tasks from the other job cannot be assigned. In order to achieve the best possible performance for both jobs, we need to consider the placement of their tasks jointly, rather than independently.

### 5.1.2 An Intuition on Task Assignment

We have illustrated two ways of assigning tasks to datacenters in Fig. 5.2 and Fig. 5.3. Intuitively, DC3 is a favorable location for tasks from both jobs, since they all have part of their input data stored in this datacenter, and the links of DC1-DC3 and DC2-DC3 both have high bandwidth. For simplicity, we assume that all the tasks are identical, with the same execution time, thus the job completion time is determined by the network transfer time. If the scheduler tries to optimize task assignment of these jobs independently, the result is shown in Fig.5.2. To optimize the assignment of job A, task tA2 would be assigned to the only available computing slot in DC3, and tA1 would be placed in DC2, which result in a network transfer time of $\max\{100/80, 200/160, 100/150\} = 1.25$ seconds. Then, if the scheduler continues to optimize the assignment of job B, DC1 and DC2 would be selected to distribute task tB1 and tB2, respectively, resulting in the transfer time of $\max\{200/80, 200/100, 300/160\} = 2.5$ seconds for job B.

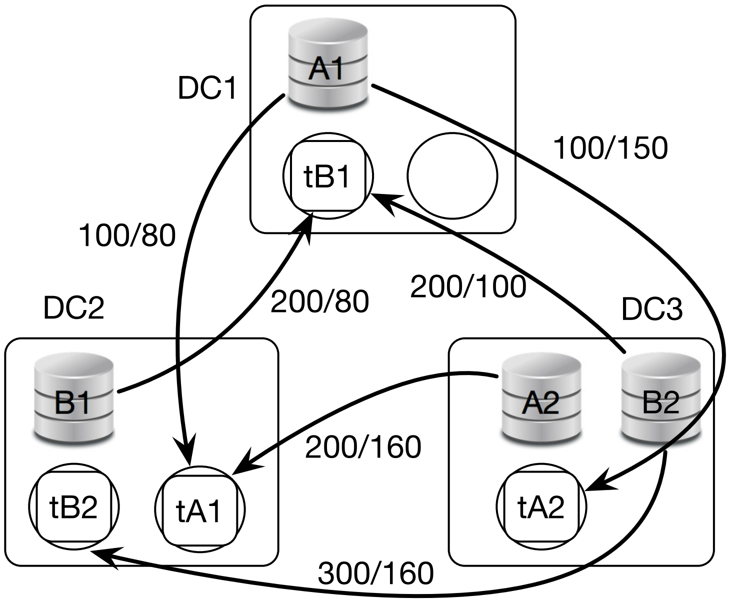


Figure 5.2: A task assignment that favors the performance of job A at the cost of job B.

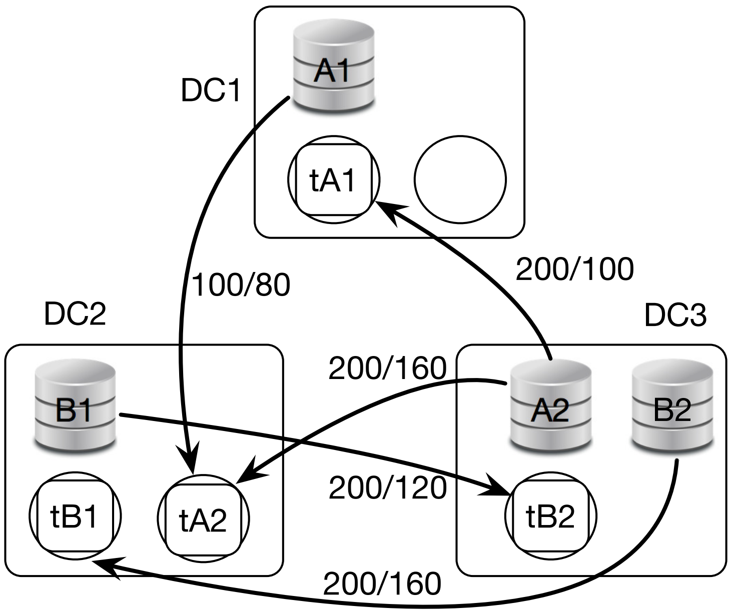


Figure 5.3: The optimal assignment for both job A and job B.

However, this placement is not optimal when considering the performance of these jobs jointly. Instead, we show the optimal assignment for both jobs satisfying max-min fairness in Fig. 5.3. With this assignment, task tB2 of job B would occupy the computing slot in DC3, which avoids the transfer of 300 MB data from dataset B2. Task tB1 is assigned to DC2 rather than DC1, which takes advantage of the high bandwidth of the DC3-DC2 link. As a result, the flow patterns are illustrated in the figure. In this assignment, the network transfer times of job A and B are $\max\{200/100, 100/80, 200/160\} = 2$ seconds, and $\max\{200/160, 200/120\} = 5/3$ seconds, respectively. Compared with the independent assignment in Fig. 5.2 where the worst performance is 2.5 seconds, this assignment results in the worst network transfer time of 2 seconds (job A), which is optimal if we wish to minimize the worst job completion time, and is fair in terms of the performance achieved by both jobs.

We are now ready to formally construct a mathematical model to study the problem of optimizing task assignment with max-min fairness across multiple jobs to be achieved.

## 5.2 Model and Formulation

We consider a set of data parallel jobs $\mathcal{K} = \{1, 2, \cdots, K\}$ submitted to the scheduler for task assignment. The input data of these jobs are distributed across a set of geo-distributed datacenters, represented by $\mathcal{D} = \{1, 2, \cdots, J\}$. Each job $k \in \mathcal{K}$ has a set of parallel tasks $\mathcal{T}_k = \{1, 2, \cdots, n_k\}$ to be launched on available computing slots in these datacenters. We use $a_j$ to denote the capacity of available computing slots in datacenter $j \in \mathcal{D}$.

For each task $i \in \mathcal{T}_k$ of job $k$, the time it takes to complete consists of both the network transfer time, denoted by $c^k_{i,j}$, to fetch the input data if the task is assigned to datacenter $j$, and the execution time represented by $e^k_{i,j}$. The network transfer time is determined by both the amount of data to be read, and the bandwidth on the link the data traverses. Let $S^k_i$ denote the set of datacenters where the input data of task $i$ from job $k$ are stored, called the source datacenters of this task for convenience. The task needs to read the input data from each of its source datacenters $s \in S^k_i$, the amount of which is represented by $d^{k,s}_i$. Let $b_{s,j}$ [1] represent the bandwidth of the link from datacenter $s$ to datacenter $j$ $(s \neq j)$. Hence, the network transfer time of task $i \in \mathcal{T}_k$, if assigned to datacenter $j$, is expressed as follows:

$$c^k_{i,j} = \begin{cases} 0, & \text{when } S^k_i = \{j\}; \\ \max\limits_{s \in S^k_i, s \neq j} d^{k,s}_i / b_{sj}, & \text{otherwise.} \end{cases} \tag{5.1}$$

This indicates that when all the input data of task $i$ are stored in the same datacenter $j$, *i.e.*, $S^k_i = \{j\}$, there would not be any traffic generated across datacenters, and thus the network transfer time is 0. Otherwise, task $i$ fetches data from each of its remote source datacenter $s \in S^k_i, s \neq j$ with the completion time of $d^{k,s}_i / b_{sj}$. As the network

[1]On popular cloud platforms (e.g., Amazon EC2 and Google Could), inter-datacenter wide-area networks are provided as a shared service, where user-generated flows will compete with millions of other flows. As a result, each inter-datacenter TCP flow will get a fair share of the link capacity. Our measurement with iperf3 on EC2 verifies this assumption.

transfer completes when input data from all the source datacenters have been fetched, the transfer time $c_{i,j}^k$ is represented as the maximum of $d_i^{k,s}/b_{sj}$ over all the remote source datacenters.

The assignment of a task is represented with a binary variable $x_{i,j}^k$, indicating whether the $i$-th task of job $k$ is assigned to datacenter $j$. A job $k$ completes when its slowest task finishes, thus the job completion time of $k$, represented by $\tau_k$, is determined by the maximum completion time among all of its tasks, expressed as follows:

$$\tau_k = \max_{i \in \mathcal{T}_k, j \in \mathcal{D}} x_{i,j}^k (c_{i,j}^k + e_{i,j}^k) \tag{5.2}$$

As the computing slots in all the datacenters are shared by tasks from multiple jobs, we would like to obtain an optimal task assignment without exceeding the resource capacities. To be more specific, our scheduler would decide the assignment of all the tasks, aiming to optimize the worst performance achieved among all the jobs with respect to their job completion times, and then optimize the next worst performance without impacting the previous one, and so on. This is executed repeatedly until the completion times have been optimized for all the jobs. Such an objective can be rigorously formulated as a *lexicographical minimization* problem, with the following definitions as its basis.

**Definition 4.** *Let $\langle \boldsymbol{v} \rangle_k$ denote the k-th ($1 \le k \le K$) largest element of $\boldsymbol{v} \in \mathbb{Z}^K$, implying $\langle \boldsymbol{v} \rangle_1 \ge \langle \boldsymbol{v} \rangle_2 \ge \cdots \ge \langle \boldsymbol{v} \rangle_K$. Intuitively, $\boldsymbol{\langle v \rangle} = (\langle \boldsymbol{v} \rangle_1, \langle \boldsymbol{v} \rangle_2, \cdots, \langle \boldsymbol{v} \rangle_K)$ represents the non-increasingly sorted version of $\boldsymbol{v}$.*

**Definition 5.** *For any $\boldsymbol{\alpha} \in \mathbb{Z}^K$ and $\boldsymbol{\beta} \in \mathbb{Z}^K$, if $\langle \boldsymbol{\alpha} \rangle_1 < \langle \boldsymbol{\beta} \rangle_1$ or $\exists k \in \{2, 3, \cdots, K\}$ such that $\langle \boldsymbol{\alpha} \rangle_k < \langle \boldsymbol{\beta} \rangle_k$ and $\langle \boldsymbol{\alpha} \rangle_i = \langle \boldsymbol{\beta} \rangle_i, \forall i \in \{1, \cdots, k-1\}$, then $\boldsymbol{\alpha}$ is* lexicographically smaller *than $\boldsymbol{\beta}$, represented as $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$. Similarly, if $\langle \boldsymbol{\alpha} \rangle_k = \langle \boldsymbol{\beta} \rangle_k, \forall k \in \{1, 2, \cdots, K\}$ or $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$, then $\boldsymbol{\alpha}$ is* lexicographically no greater *than $\boldsymbol{\beta}$, represented as $\boldsymbol{\alpha} \preceq \boldsymbol{\beta}$.*

**Definition 6.** *$\underset{\boldsymbol{x}}{\text{lexmin}}\, \boldsymbol{f}(\boldsymbol{x})$ represents the* lexicographical minimization *of the vector $\boldsymbol{f} \in \mathbb{R}^N$, which consists of $N$ objective functions of $\boldsymbol{x}$. To be particular, the optimal solution*

*$\boldsymbol{x}^* \in \mathbb{R}^K$ achieves the optimal $\boldsymbol{f}^*$, in the sense that $\boldsymbol{f}^* = \boldsymbol{f}(\boldsymbol{x}^*) \preceq \boldsymbol{f}(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathbb{R}^K$.*

With these definitions, we are now ready to formulate our optimal task assignment problem among sharing jobs as follows:

$$\underset{\boldsymbol{x}}{\text{lexmin}} \quad \boldsymbol{f} = (\tau_1, \tau_2, \cdots, \tau_K) \tag{5.3}$$

$$\text{s.t.} \quad \tau_k = \max_{i \in \mathcal{T}_k, j \in \mathcal{D}} x_{i,j}^k (c_{i,j}^k + e_{i,j}^k), \forall k \in \mathcal{K} \tag{5.4}$$

$$\sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{T}_k} x_{i,j}^k \leq a_j, \quad \forall j \in \mathcal{D} \tag{5.5}$$

$$\sum_{j \in \mathcal{D}} x_{i,j}^k = 1, \quad \forall i \in \mathcal{T}_k, \ \forall k \in \mathcal{K} \tag{5.6}$$

$$x_{i,j}^k \in \{0, 1\}, \quad \forall i \in \mathcal{T}_k, \ \forall j \in \mathcal{D}, \ \forall k \in \mathcal{K} \tag{5.7}$$

where constraint (5.4) represents the completion time of each job $k$ as aforementioned. Constraint (5.5) indicates that the total number of tasks to be assigned to datacenter $j$ does not exceed its capacity $a_j$, which is the total number of available computing slots. Constraint (5.6) implies that each task should be assigned to a single datacenter.

The objective is a vector $\boldsymbol{f} \in \mathbb{R}^K$ with $K$ elements, each standing for the completion time of a particular job $k \in \mathcal{K}$. According to the previous definitions, the optimal $\boldsymbol{f}^*$ is lexicographically no greater than any $\boldsymbol{f}$ obtained with a feasible assignment, which means that when sorting them in a non-increasing order, if their $k$-th largest element satisfies $\langle \boldsymbol{f}^* \rangle_{k'} = \langle \boldsymbol{f} \rangle_{k'}, \forall k' < k$ and $\langle \boldsymbol{f}^* \rangle_k \neq \langle \boldsymbol{f} \rangle_k$, then we have $\langle \boldsymbol{f}^* \rangle_k < \langle \boldsymbol{f} \rangle_k$. This implies that the first largest element of $\boldsymbol{f}^*$, *i.e.*, the slowest completion time, is the minimum among all $\boldsymbol{f}$. Then among all $\boldsymbol{f}$ with the same worst completion time, the second worst completion time in $\boldsymbol{f}^*$ is the minimum, and so on. In this way, solving this problem would result in an optimal assignment vector $\boldsymbol{x}^*$, with which all the job completion times are minimized.

## 5.3 Optimizing the Worst Completion Time among Concurrent Jobs

Problem (5.3) is a vector optimization with multiple objectives. In this section, we consider the single-objective subproblem of optimizing the worst job performance as follows:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \max_{k\in\mathcal{K}} \,(\tau_k) \\ \text{s.t.} \quad & \text{Constraints } (5.4), (5.5), (5.6) \text{ and } (5.7). \end{aligned} \tag{5.8}$$

which is a primary step towards solving the original problem, to be elaborated in the next section.

Substituting the completion time $\tau_k$ in the objective with the expression in constraint (5.4), we have the following problem with the non-linear constraint (5.4) eliminated:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \max_{k\in\mathcal{K}} \,(\max_{i\in\mathcal{T}_k, j\in\mathcal{D}} x^k_{i,j}(c^k_{i,j}+e^k_{i,j})) \\ \text{s.t.} \quad & \text{Constraints } (5.5), (5.6) \text{ and } (5.7). \end{aligned} \tag{5.9}$$

Though this problem is an integer programming problem, we will show that it can be transformed into an equivalent linear programming (LP) problem after an in-depth investigation of its structure. As a result of such a transformation, it can be solved efficiently to obtain the optimal schedule vector $\boldsymbol{x}$. Our transformation takes advantage of its features of *separable convex objective* and *totally unimodular linear constraints*, and it involves three major steps to be elaborated in the following subsections.

### 5.3.1 Separable Convex Objective

In the first step, we will show that the optimal solution for Problem (5.9) can be obtained by solving a problem with a separable convex objective function, which is represented as a summation of convex functions with respect to each single variable $x^k_{i,j}$.

We first show that the optimal solution of Problem (5.9) can be obtained by solving the following problem:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{lexmin}} \quad & \boldsymbol{g} = (\phi(x_{1,1}^1), \cdots, \phi(x_{i,j}^k), \cdots, \phi(x_{n_K,J}^K)) \\ \text{s.t.} \quad & \text{Constraints (5.5), (5.6) and (5.7).} \end{aligned}$$

where $\phi(x_{i,j}^k) = x_{i,j}^k(c_{i,j}^k + e_{i,j}^k), \forall i \in \mathcal{T}_k, \ \forall j \in \mathcal{D}, \ \forall k \in \mathcal{K}$, and $\boldsymbol{g}$ is a vector with the dimension of $M = |\boldsymbol{g}| = J\sum_{k=1}^{K} n_k$. For this problem, the objective includes minimizing the maximum element in $\boldsymbol{g}$, which is the worst completion time across all the jobs. Therefore, the optimal assignment variables $\boldsymbol{x}^*$ that gives $\boldsymbol{g}^*$ is also the optimal solution for Problem (5.9).

Let $\varphi(\boldsymbol{g})$ define a function of $\boldsymbol{g}$:

$$\varphi(\boldsymbol{g}) = \sum_{m=1}^{|\boldsymbol{g}|} |\boldsymbol{g}|^{g_m} = \sum_{m=1}^{M} M^{g_m}$$

where $g_m$ is the $m$-th element of $\boldsymbol{g}$.

**Lemma 4.** *$\varphi(\cdot)$ preserves the order of* lexicographically no greater *($\preceq$), i.e., $\boldsymbol{g}(\boldsymbol{x}^*) \preceq \boldsymbol{g}(\boldsymbol{x}) \iff \varphi(\boldsymbol{g}(\boldsymbol{x}^*)) \leq \varphi(\boldsymbol{g}(\boldsymbol{x}))$.*

*Proof.* We first consider $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{Z}^K$ that satisfies $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$. If we use the integer $\tilde{k}(1 \leq \tilde{k} \leq K)$ to represent the first non-zero element of $\langle\boldsymbol{\alpha}\rangle - \langle\boldsymbol{\beta}\rangle$, we have $\langle\boldsymbol{\alpha}\rangle_k = \langle\boldsymbol{\beta}\rangle_k, \forall\tilde{k} \leq k \leq K$ and $\langle\boldsymbol{\alpha}\rangle_{\tilde{k}} < \langle\boldsymbol{\beta}\rangle_{\tilde{k}}$. Assume $\langle\boldsymbol{\alpha}\rangle_{\tilde{k}} = m$, then $\langle\boldsymbol{\beta}\rangle_{\tilde{k}} \geq m + 1$.

$$\begin{aligned}
\varphi(\boldsymbol{\alpha}) &= \sum_{k=1}^{K} K^{\langle\boldsymbol{\alpha}\rangle_k} = \sum_{k=1}^{\tilde{k}-1} K^{\langle\boldsymbol{\alpha}\rangle_k} + K^{\langle\boldsymbol{\alpha}\rangle_{\tilde{k}}} + \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\alpha}\rangle_k} \\
&\leq \sum_{k=1}^{\tilde{k}-1} K^{\langle\boldsymbol{\alpha}\rangle_k} + K^{\langle\boldsymbol{\alpha}\rangle_{\tilde{k}}} + (K-\tilde{k})K^{\langle\boldsymbol{\alpha}\rangle_{\tilde{k}}} \\
&= \sum_{k=1}^{\tilde{k}-1} K^{\langle\boldsymbol{\alpha}\rangle_k} + (K+1-\tilde{k}) \cdot K^{\langle\boldsymbol{\alpha}\rangle_{\tilde{k}}} \\
&< \sum_{k=1}^{\tilde{k}-1} K^{\langle\boldsymbol{\alpha}\rangle_k} + K \cdot K^m,
\end{aligned}$$

where the first inequality holds as $\langle\boldsymbol{\alpha}\rangle_{\tilde{k}} \geq \langle\boldsymbol{\alpha}\rangle_k, \forall \tilde{k}+1 \leq k \leq K$.

$$\begin{aligned}
\varphi(\boldsymbol{\beta}) &= \sum_{k=1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k} = \sum_{k=1}^{\tilde{k}-1} K^{\langle\boldsymbol{\beta}\rangle_k} + K^{\langle\boldsymbol{\beta}\rangle_{\tilde{k}}} + \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k} \\
&> \sum_{k=1}^{\tilde{k}-1} K^{\langle\boldsymbol{\beta}\rangle_k} + K^{\langle\boldsymbol{\beta}\rangle_{\tilde{k}}} + (K-\tilde{k}) \cdot 0 \\
&\geq \sum_{k=1}^{\tilde{k}-1} K^{\langle\boldsymbol{\beta}\rangle_k} + K \cdot K^m.
\end{aligned}$$

Given that $\sum_{k=1}^{\tilde{k}-1} K^{\langle\boldsymbol{\alpha}\rangle_k} = \sum_{k=1}^{\tilde{k}-1} K^{\langle\boldsymbol{\beta}\rangle_k}$, we have proved that $\varphi(\boldsymbol{\alpha}) < \varphi(\boldsymbol{\beta})$.

If $\boldsymbol{\alpha} = \boldsymbol{\beta}$, which means that $\langle\boldsymbol{\alpha}\rangle_k = \langle\boldsymbol{\beta}\rangle_k, \forall 1 \leq k \leq K$, it is trivially true that $\varphi(\boldsymbol{\alpha}) = \sum_{k=1}^{K} K^{\langle\boldsymbol{\alpha}\rangle_k} = \sum_{k=1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k} = \varphi(\boldsymbol{\beta})$. Thus, we have proved $\boldsymbol{\alpha} \preceq \boldsymbol{\beta} \Longrightarrow \varphi(\boldsymbol{\alpha}) \leq \varphi(\boldsymbol{\beta})$.

We further prove $\varphi(\boldsymbol{\alpha}) \leq \varphi(\boldsymbol{\beta}) \Longrightarrow \boldsymbol{\alpha} \preceq \boldsymbol{\beta}$ by proving its contrapositive: $\neg(\boldsymbol{\alpha} \preceq \boldsymbol{\beta}) \Longrightarrow \varphi(\boldsymbol{\alpha}) > \varphi(\boldsymbol{\beta})$. $\neg(\boldsymbol{\alpha} \preceq \boldsymbol{\beta})$ implies $\boldsymbol{\alpha} \neq \boldsymbol{\beta}$ and the first non-zero element of $\langle\boldsymbol{\alpha}\rangle - \langle\boldsymbol{\beta}\rangle$ is positive, which further indicates the first non-zero element of $\langle\boldsymbol{\beta}\rangle - \langle\boldsymbol{\alpha}\rangle$ is negative, *i.e.*, $\boldsymbol{\beta} \prec \boldsymbol{\alpha}$. Thus, the contrapositive is equivalent to $\boldsymbol{\beta} \prec \boldsymbol{\alpha} \Longrightarrow \varphi(\boldsymbol{\beta}) < \varphi(\boldsymbol{\alpha})$, which has already been proved previously using the exchanged notations of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

With $\boldsymbol{\alpha} \preceq \boldsymbol{\beta} \iff \varphi(\boldsymbol{\alpha}) \leq \varphi(\boldsymbol{\beta})$ holding for any $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ of the same dimension, we complete the proof by letting $\boldsymbol{\alpha} = \boldsymbol{g}(\boldsymbol{x}^*)$ and $\boldsymbol{\beta} = \boldsymbol{g}(\boldsymbol{x})$. □

Based on Lemma 4, we have

$$\operatorname*{lexmin}_{\boldsymbol{x}} \ \boldsymbol{g} \iff \min_{\boldsymbol{x}} \ \ \varphi(\boldsymbol{g}) = \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{T}_k} \sum_{j\in\mathcal{D}} M^{\phi(x_{i,j}^k)}$$

where the objective function $\varphi(\boldsymbol{g})$ is a summation of the term $M^{\phi(x_{i,j}^k)}$, which is a convex function of the single variable $x_{i,j}^k$.

Therefore, solving Problem (5.9) is equivalent to solving the following problem with a separable convex objective:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{T}_k} \sum_{j\in\mathcal{D}} M^{\phi(x_{i,j}^k)} \\ \text{s.t.} \quad & \text{Constraints } (5.5), (5.6) \text{ and } (5.7). \end{aligned} \tag{5.10}$$

## 5.3.2 Totally Unimodular Linear Constraints

In the second step, we investigate the coefficient matrix of linear constraints (5.5) and (5.6). Recall the definition of totally unimodular matrix in Sec. 4.3, we have the following lemma.

**Lemma 5.** *The coefficients of constraints (5.5) and (5.6) form a totally unimodular matrix.*

*Proof.* Let $\mathbf{A}_{m\times n}$ denote the coefficient matrix of all the linear constraints (5.5) and (5.6), where $m = J + \sum_{k=1}^{K} n_k$, representing the total number of the constraints, and $n = J\sum_{k=1}^{K} n_k$, denoting the dimension of the variable $\boldsymbol{x}$.

It is obvious that any element of $\mathbf{A}_{m\times n}$ is either 0 or 1, satisfying the first condition. For any row subset $R \subset \{1, 2, \cdots, m\}$, we can select all the elements that belong to $\{1, 2, \cdots, J\}$ to form the set $R_1$. As such, $R$ is divided into two disjoint sets, $R_1$ and $R_2 = R - R_1$. It is easy to check that for coefficient matrix of constraint (5.5), the summation of all its rows, represented by rows $\{1, 2, \cdots, J\}$, is a $1 \times n$ vector with all the elements equal to 1. Similarly, for coefficient matrix of constraint (5.6), the summation of all its rows, represented by rows $\{J+1, J+2, \cdots, J+\sum_{k=1}^{K} n_k\}$, is also a $1 \times n$ vector whose elements are 1. Hence, we can easily derive that $\sum_{i \in R_1} a_{ij} \leq 1, \sum_{i \in R_2} a_{ij} \leq 1, \forall j \in \{1, 2, \cdots, n\}$. Eventually, we have $|\sum_{i \in R_1} a_{ij} - \sum_{i \in R_2} a_{ij}| \leq 1, \forall j \in \{1, 2, \cdots, n\}$, and the second condition is satisfied.

In summary, we have shown that both conditions for total unimodularity are satisfied, thus the coefficient matrix $\mathbf{A}_{m \times n}$ is totally unimodular. $\square$

### 5.3.3 Structure-Inspired Equivalent LP Transformation

In the final step, exploiting the problem structure of totally unimodular constraints and separable convex objective, we can use the $\lambda$-representation technique to transform Problem (5.10) to a linear programming problem that has the same optimal solution.

For a single integer variable $y \in \mathcal{Y} = \{0, 1, \cdots, Y\}$ , the convex function $h : \mathcal{Y} \rightarrow \mathbb{R}$ can be linearized with the $\lambda$-representation as follows:

$$h(y) = \sum_{s \in \mathcal{Y}} h(s)\lambda_s, \quad y = \sum_{s \in \mathcal{Y}} s\lambda_s$$

$$\sum_{s \in \mathcal{Y}} \lambda_s = 1, \quad \lambda_s \in \mathbb{R}^+, \ \forall s \in \mathcal{Y}.$$

In our problem, we apply the $\lambda$-representation technique to each convex function $h_{i,j}^k(x_{i,j}^k) = M^{\phi(x_{i,j}^k)} : \{0, 1\} \rightarrow \mathbb{R}$ as follows:

$$h_{i,j}^k(x_{i,j}^k) = \sum_{s \in \{0,1\}} M^{s(c_{i,j}^k + e_{i,j}^k)} \lambda_{i,j}^{k,s} = \lambda_{i,j}^{k,0} + M^{c_{i,j}^k + e_{i,j}^k} \lambda_{i,j}^{k,1}$$

which removes the variable $x_{i,j}^k$ by sampling at each of its possible value $s \in \{0, 1\}$, weighted by the newly introduced variables $\lambda_{i,j}^{k,s} \in \mathbb{R}^+, \forall s \in \{0,1\}$ that satisfy

$$x_{i,j}^k = \sum_{s\in\{0,1\}} s\lambda_{i,j}^{k,s} = \lambda_{i,j}^{k,1}$$

$$\sum_{s\in\{0,1\}} \lambda_{i,j}^{k,s} = \lambda_{i,j}^{k,0} + \lambda_{i,j}^{k,1} = 1$$

Further, with linear relaxation on the integer constraints (5.7), we obtain the following linear programming problem:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{\lambda}} \quad & \sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{T}_k}\sum_{j\in\mathcal{D}} (\lambda_{i,j}^{k,0} + M^{c_{i,j}^k + e_{i,j}^k}\lambda_{i,j}^{k,1}) \qquad (5.11) \\ \text{s.t.} \quad & x_{i,j}^k = \lambda_{i,j}^{k,1}, \ \ \forall k \in \mathcal{K}, i \in \mathcal{T}_k, j \in \mathcal{D} \\ & \lambda_{i,j}^{k,0} + \lambda_{i,j}^{k,1} = 1, \ \ \forall k \in \mathcal{K}, i \in \mathcal{T}_k, j \in \mathcal{D} \\ & \lambda_{i,j}^{k,0}, \lambda_{i,j}^{k,1}, x_{i,j}^k \in \mathbb{R}^+, \ \forall k \in \mathcal{K}, i \in \mathcal{T}_k, j \in \mathcal{D} \\ & \text{Constraints (5.5) and (5.6).} \end{aligned}$$

**Theorem 3.** *An optimal solution to Problem (5.11) is an optimal solution to Problem (5.8).*

*Proof.* The property of total unimodularity ensures that an optimal solution to the relaxed LP problem (5.11) has integer values of $x_{i,j}^k$, which is an optimal solution to Problem (5.10), and thus an optimal solution to Problem (5.9) as demonstrated in Sec. 5.3.1. Moreover, Problem (5.8) and (5.9) are equivalent forms, completing the proof. □

Therefore, the optimal assignment that minimizes the worst completion time among all the jobs can be obtained by solving Problem (5.11) with efficient LP solvers, such as MOSEK [17].

## 5.4 Iteratively Optimizing Worst Completion Times to Achieve Max-Min Fairness

With the subproblem of minimizing the worst completion time efficiently solved as an LP problem (5.11), we continue to solve our original multi-objective problem (5.3) by minimizing the next worst completion time repeatedly.

After solving the subproblem, it is known that the optimal worst completion time is achieved by job $k^*$, whose slowest task $i^*$ is assigned to datacenter $j^*$. We then fix the computed assignment of the slowest task of job $k^*$, which means that the corresponding schedule variable $x_{i^*,j^*}^{k^*}$ is removed from the variable set $\boldsymbol{x}$ for the next round. Also, since task $i^*$ is to be assigned to datacenter $j^*$, it is intuitive that all the assignment variables associated with it should be fixed as zero and removed from $\boldsymbol{x}$: $x_{i^*,j}^{k^*} = 0, \forall j \neq j^*, j \in \mathcal{D}$.

As we have fixed a part of the assignment, the resource capacities should be updated in our problem constraints in the next round. For example, if $x_{i^*,j^*}^{k^*} = 1$, which means that the $i^*$th task of job $k^*$ would be assigned to datacenter $j^*$, then for the problem in the next round, $x_{i^*,j^*}^{k^*}$ is no longer a variable. The resource capacity constraints should be updated as $\sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{T}_k, (k,i) \neq (k^*,i^*)} x_{i,j^*}^{k} \leq a_{j^*} - 1$.

Moreover, the completion time of job $k^*$ is obtained as $\phi(x_{i^*,j^*}^{k^*})$, the completion time of its slowest task $i^*$ assigned to datacenter $j^*$, yet the assignment of other tasks has not been fixed, which would be the variables $(x_{i,j}^{k^*}, \forall (i,j) \neq (i^*,j^*))$ of the problem in the next round. We set the associated completion times of these variables as $x_{i,j}^{k^*}(c_{i^*,j^*}^{k^*} + e_{i^*,j^*}^{k^*})$. The rationale is that no matter how fast other tasks of $k^*$ complete, the completion time is determined by the slowest task $i^*$. This ensures that the completion time optimized in the next round is achieved by another job, rather than a task of job $k^*$ (other than its slowest). This is better illustrated with the example in Fig. 5.4.

In this example, after the calculation in the first round, task `tA1` assigned in `DC2` achieves the worst completion time of 4s, among the two sharing jobs. In the second round, if we do not change the associated completion time for another task `tA2`, assign-

**Algorithm 4:** Performance-Optimal Task Assignment among Jobs with Max-Min Fairness.

```
Input:
    Input data sizes $d_i^{k,s}$ and link bandwidth $b_{sj}$ to obtain network transfer time $c_{i,j}^k$ (by Eq. 5.1); execution time $e_{i,j}^k$; datacenter resource capacity $a_j$;
Output:
    Task assignment $x_{i,j}^k, \forall k \in \mathcal{K}, \forall i \in \mathcal{T}_k, \forall j \in \mathcal{D}$;
1: Initialize $\mathcal{K}' = \mathcal{K}$;
2: while $\mathcal{K}' \neq \varnothing$ do
3:     Solve the LP Problem (5.11) to obtain the solution $\boldsymbol{x}$;
4:     Obtain $x_{k^*,i^*}^{j^*} = \operatorname*{argmax}_{x_{i,j}^k \in \boldsymbol{x}} \phi(x_{i,j}^k)$;
5:     Fix $x_{i^*,j}^{k^*}, \forall j \in \mathcal{D}$; remove them from variable set $\boldsymbol{x}$;
6:     Update the corresponding resource capacities in Constraints (5.5);
7:     Set $\phi(x_{i,j}^{k^*}) = x_{i,j}^{k^*}(c_{i^*,j^*}^{k^*} + e_{i^*,j^*}^{k^*}),\ \forall i \in \mathcal{T}_{k^*}, \forall j \in \mathcal{D}$;
8:     Remove $k^*$ from $\mathcal{K}'$;
9: end while
```

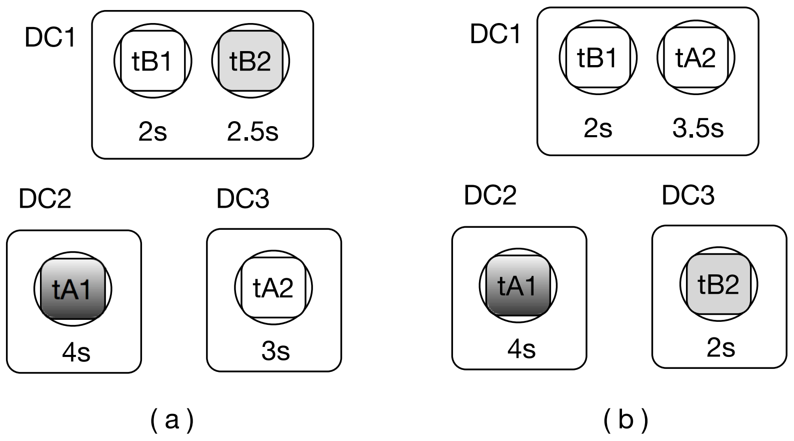


Figure 5.4: Two possible assignments of tasks from two jobs.

ment (a) would be calculated as the optimal solution, since it achieves the completion times of $(4, 3, 2.5, 2)$ for the four tasks, which is lexicographically smaller than $(4, 3.5, 2, 2)$ achieved by assignment (b). However, assignment (a) minimizes the next worst completion time for another task of the same job A, which does not match our original objective to optimize the worst completion time for job B. Instead, if we set the associated completion times of all job A's tasks as 4, the objective function achieved by assignment (b) would be $(4, 4, 2, 2)$, better than $(4, 4, 2.5, 2)$ given by assignment (a). In this way, the optimization can correctly choose assignment (b) to achieve the optimal completion time of job `B`. Note that although the completion time of task `tA2` in assignment (b) is longer than in assignment (a), the completion time of job `A` remains the same, which is 4s.

As a result, the subproblem in the next round is solved over a decreased set of variables with updated constraints and objectives, so that the next worst job completion time would be optimized, without impacting the worst job performance in this round. Such a procedure is repeatedly executed until the last worst completion time of jobs has been optimized, and the max-min fairness has been achieved, as summarized in Algorithm 4.

## 5.5 Performance Evaluation

Having proved the theoretical optimality and efficiency of our scheduling solution, we proceed to implement it in Apache Spark, and demonstrate its effectiveness in optimizing job completion times in real-world experiments. Moreover, we present an extensive array of large-scale simulation results to demonstrate the effectiveness of our algorithm in achieving max-min fair job performance.

### 5.5.1 Real-World Implementation and Experiments

#### Design and Implementation

In Apache Spark [2], a job can be represented by a Directed Acyclic Graph (DAG), where each node represents a task and each directed edge indicates a precedence constraint. In general, the problem of assigning all the tasks in a DAG to a number of worker nodes, with the objective of minimizing the job completion time, is known as NP-Complete [66]. As a practical and efficient design, Spark schedules tasks stage by stage, which is handled by the *DAG scheduler*. When a job is submitted, it is transformed into a DAG of tasks, categorized into a set of stages. The DAG scheduler will then submit the tasks within each stage, called a `TaskSet`, to the task scheduler whenever the stage is ready to be scheduled, implying that all its parent stages have completed.

Fortunately, the task scheduler in Spark has access to most of the information that

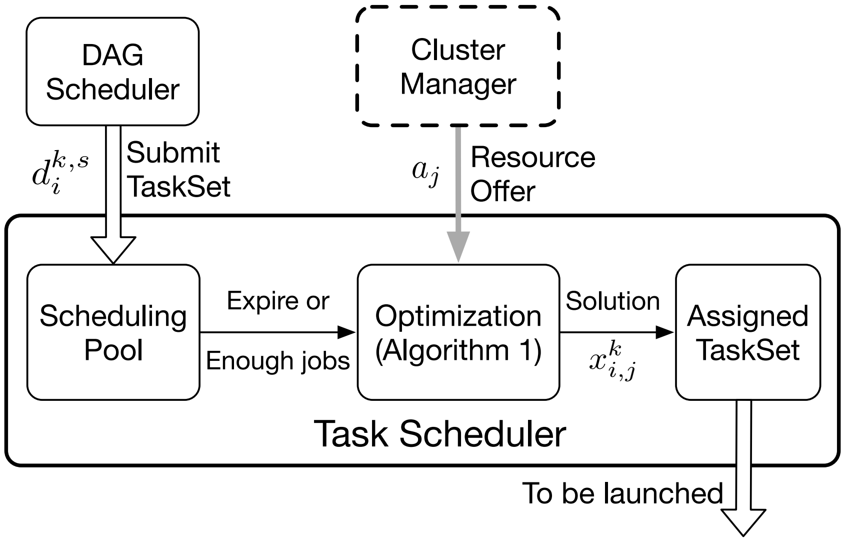


Figure 5.5: The implementation of our task assignment algorithm in Spark.

our algorithm needs as its input. As a result, we have implemented our new task assignment algorithm as an extension to Spark's `TaskScheduler` module. The design of our implementation is illustrated in Fig. 5.5. In our implementation, as soon as a `TaskSet` in a job has been submitted by the DAG scheduler to the task scheduler, they will be immediately queued in the scheduling pool. With a number of concurrent jobs waiting to be scheduled, our algorithm will be triggered when a preset timer expires or when the number of pending jobs in the pool exceeds a certain threshold.

To optimize the assignment of tasks, our algorithm requires knowledge about the size and location of the output data from each `map` task, represented by $d_i^{k,s}$ in our formulation. Such knowledge can be obtained from the `MapOutputTracker`, which are further saved in the `TaskSet` of `reduce` tasks. The available bandwidth ($b_{sj}$) between each pair of datacenters ($s$ to $j$) can be measured with the `iperf2` utility. The task execution time $e_{i,j}^k$ can be obtained from historical data, which is the common practice for recurrent jobs. (We can also utilize advanced prediction techniques such as [67] to estimate the task execution time in our future work.) In addition, information about the amount of available resources, corresponding to $a_j$ in our formulation, can be obtained from the cluster manager. Now that all the input required by our algorithm is ready, an optimal

Table 5.1: Available bandwidth across geo-distributed datacenters (Mbps).

| | Virginia | Oregon | Ireland | Sing | Sydney | SP |
|---|---|---|---|---|---|---|
| Virginia | 1000 | 169 | 154 | 52 | 53 | 104 |
| Oregon | - | 1000 | 71 | 69 | 77 | 68 |
| Ireland | - | - | 1000 | 49 | 40 | 65 |
| Singapore | - | - | - | 1000 | 58 | 35 |
| Sydney | - | - | - | - | 1000 | 38 |
| San Paulo | - | - | - | - | - | 1000 |

“Sing" is short for “Singapore", and “SP" is short for “San Paulo".

assignment will be computed for all the tasks in the scheduling pool, through iteratively formulating and solving updated versions of linear programming problems, solved by the LP solver in the `Breeze` optimization library [65].

After the assignment has been computed, it will be recorded in the corresponding `TaskSet`, overriding the original task assignment preferences. When the tasks are finally submitted for execution, these assignment preferences will be satisfied in a greedy manner. Since the scheduling decisions will satisfy resource constraints by considering *Resource Offers*, each task in the `TaskSet` can be launched in any available computing slot in the assigned datacenter.

### Experimental Setup

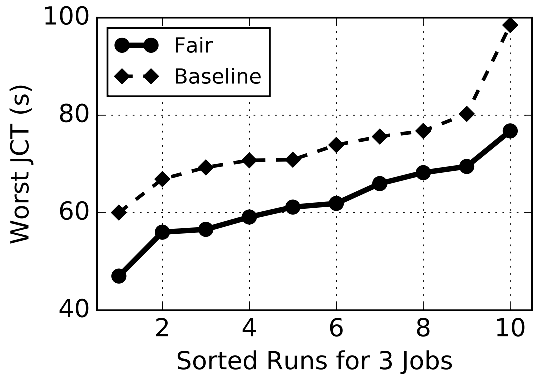
We are now ready to evaluate our real-world implementation with an extensive set of experiments deployed across 6 datacenters in Amazon EC2, located in a geographically distributed fashion across different continents. The available bandwidth between each pair of datacenters, measured with the `iperf2` utility and averaged over 5 measurements over the period of 20 minutes, is shown in Table 5.1. Compared to the intra-datacenter network where the available bandwidth is around 1 Gbps, bandwidth on inter-datacenter links are much more limited: almost all inter-continental links have less than 100 Mbps of available bandwidth. This confirms the observation that transferring large volumes of data across datacenters is likely to be time-consuming.

In our experiments, we have used a total of 12 on-demand Virtual Machine (VM) instances as Spark workers in our Spark cluster, located across 6 datacenters. Two special VM instances in Virginia (`us-east-1`) have been used as the Spark master node and the Hadoop File System (HDFS) [68] Name Node, respectively. All instances are of type `m3.large`, each with 2 vCPUs, 7.5 GB of memory, and a 32GB Solid-State Drive. In each instance, we run Ubuntu Server 14.04 LTS 64-bit (HVM), with Java 1.8 and Scala 2.11.8 installed. Hadoop 2.6.4 is installed to provide HDFS support for Spark. Our own implementation of the task assignment algorithm is based on Spark 1.6.1, a recent release as of July 2016. Our Spark cluster runs in the standalone mode, with all configurations left as default. No external resource manager (*e.g.,* YARN) or database system is activated.

In order to illustrate the efficiency of our task assignment algorithm, we use the legacy `Sort` application as the benchmark workload. We choose this workload because it is simple but primitive. As one of the simplest MapReduce applications, `Sort` has only one map and one reduce stage. However, its `sortByKey()` operation is a basic building block for many complex data analytics applications, especially in Spark SQL. It triggers an all-to-all shuffle, which introduces heavier cross-node traffic than other reduce operations such as `reduceByKey()`.

In our experiments, we have implemented a `Sort` application with multiple jobs, and submitted it to Spark for execution. The jobs are submitted to Spark in parallel threads, triggering concurrent jobs to share the resources in the cluster. Therefore, the task assignment decisions for these concurrent jobs will be made and enforced by our implementation in the `TaskScheduler`. To evaluate the performance under different workloads, we run the workload with 3, 4 and 5 concurrent jobs as separate experiments.

For each `Sort` job in our benchmark application, the default parallelism is set to 3. In other words, the job will trigger 3 reduce tasks to sort the input dataset, which has 3 partitions distributed on 3 randomly chosen worker nodes. The input dataset is



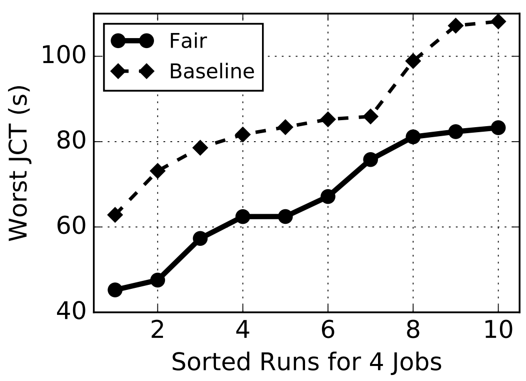


Figure 5.6: The worst job completion time in a set of (3 and 4) concurrent jobs.

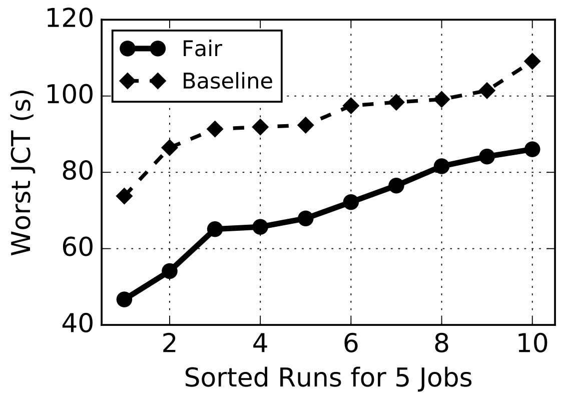
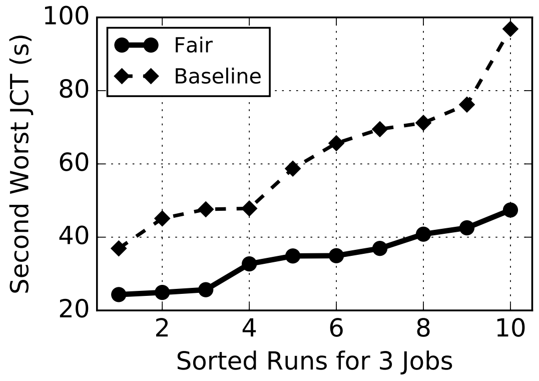


Figure 5.7: The worst job completion time in 5 concurrent jobs.

prepared as a step in the `map` task. Each partition of the generated dataset is 100 MB in size, containing 10,000 key-value pairs. Then, as the start of the `reduce` task, these key-values will be shuffled over the network. Since each datacenter has only two workers, a fraction of the shuffled traffic will be sent over inter-datacenter network links, which are likely to be the performance bottleneck. Our task assignment algorithm is specifically designed to mitigate the negative effects of such bottlenecks.

## Experimental Results

We conducted three groups of experiments, with 3, 4, and 5 concurrent jobs, respectively. In the first two groups, each job has three tasks; while in the third group, the total number



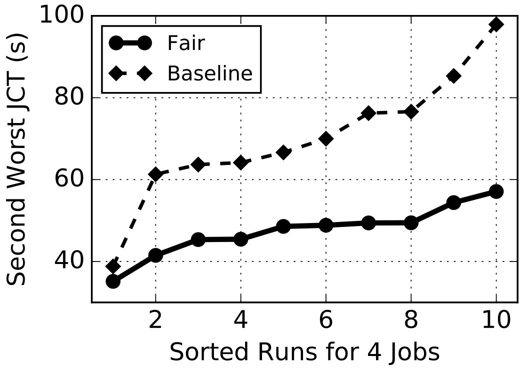


Figure 5.8: The second-worst job completion time in a set of (3 and 4) concurrent jobs.

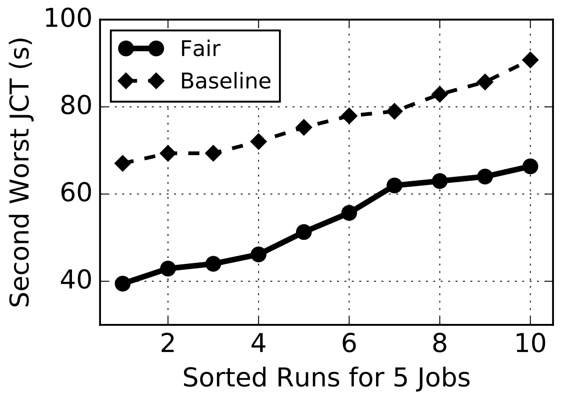


Figure 5.9: The second-worst job completion time in 5 concurrent jobs.

of tasks of the five jobs is set as the total number of available computing slots, which is 12 in our Spark cluster. In each experiment, the job completion times achieved with our optimal task assignment algorithm is compared with that achieved with the default scheduling in Spark, used as the baseline in our comparison study.

The results of our experiments are presented in Fig. 5.6, Fig. 5.7, Fig. 5.8 and Fig. 5.9, showing the worst completion times and the second worst completion times among concurrent jobs, respectively. With respect to the worst completion time, it is easy to see that our algorithm always performs better than the baseline in Spark, with a performance improvement of up to 66%, as shown in Fig. 5.6 and Fig. 5.7. With respect to the second worst completion time, our algorithm does not theoretically guarantee that it is smaller than that achieved with any unfair placement. Even without guarantees, our algorithm always shows better performance than the baseline in Spark. These experiments have shown convincing evidence that our algorithm — optimizing for max-min fairness — is effective in maximizing the worst completion time and achieving best possible performance for all concurrent jobs.

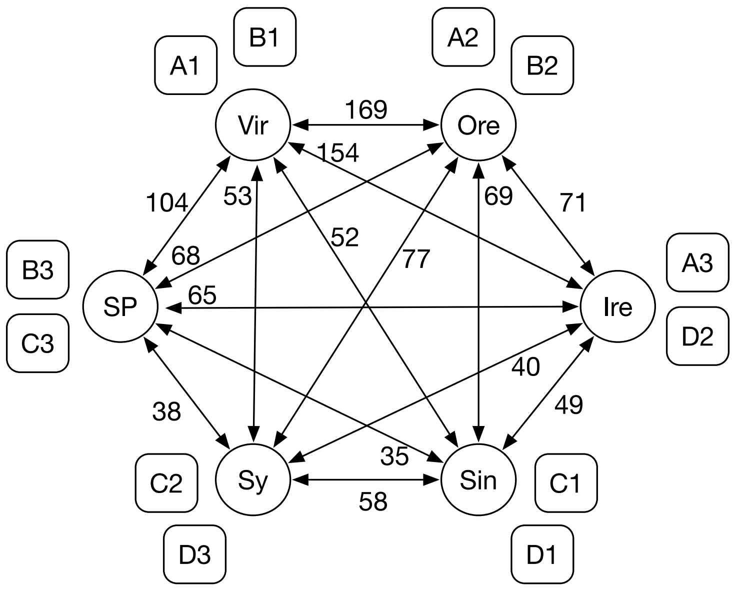


Figure 5.10: The location of input data for four jobs across six datacenters.

To offer a more in-depth examination and show why such a performance improvement can be achieved, we consider the 7-th run in the second group of our experiment, with the sharing relationship and bandwidth shown in Fig. 5.10. The six circles represent the six datacenters used in our experiment. A1 is located in the Virginia datacenter, representing the data required by tasks from job A. For each task in job A to be scheduled, a fraction of data needs to be read from all the three datasets (A1, A2 and A3). As each datacenter has two available computing slots, the assignment of 12 tasks from four jobs becomes a one-to-one mapping. In such a limited resource scenario, the assignment of a task is tightly coupled with each other. It becomes more difficult for the default strategy in Spark to find a good assignment, without optimizing across all the tasks. Moreover, the wide range of available bandwidth between datacenters is not taken into consideration

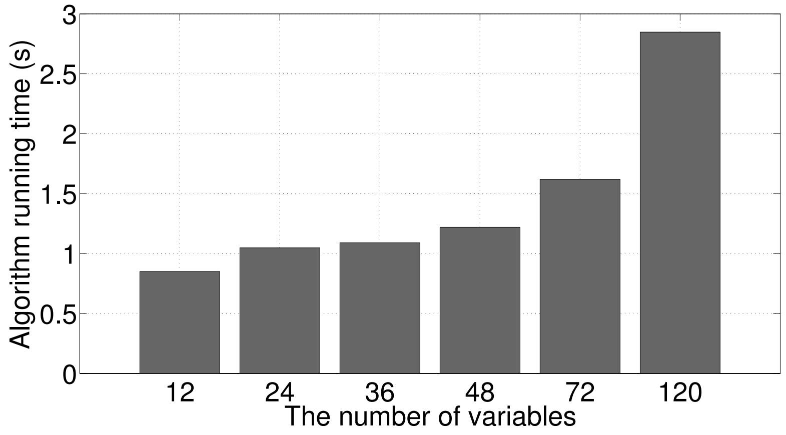


Figure 5.11: The computation times of our algorithm at different scales.

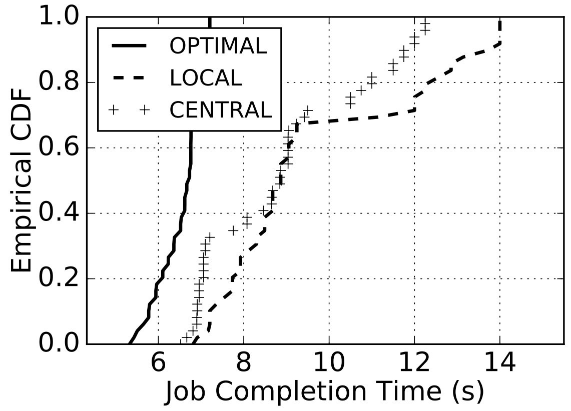
by the default scheduling in Spark, which also explains the performance improvement of our strategy over the baseline.

Furthermore, to evaluate the practicality of our algorithm, we have recorded the time it takes to calculate optimal solutions. Fig. 5.11 illustrates the computation times, each averaged over 10 runs, with the number of variables varying from 12 to 120. The linear program in our algorithm is efficient, as it takes about 1 second to obtain the solution for 48 variables. The computation time is less than 3 seconds for 120 variables, which is acceptable compared with the transfer times across datacenters that could be tens or hundreds of seconds. In our experiment, the algorithm is running in the VM with 2 vCPUs. We envision that with more powerful servers for the scheduler in a production environment, the running time could be even smaller.

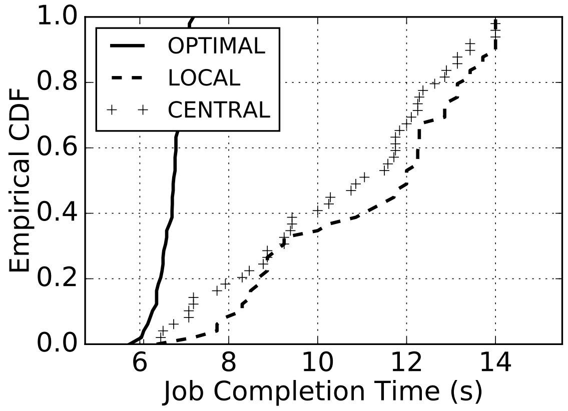
### 5.5.2 Large-Scale Simulations

We further present the results and analysis of our extensive simulations on our scheduling algorithm, to evaluate its effectiveness and performance at a finer granularity.

To allow simulations at a large scale, we implemented our simulator in `JAVA`, using the efficient `CPLEX` optimizer [69] to solve our LP problems. We conducted an extensive set of simulations, each with 50 and 100 concurrent jobs, respectively. The setting of the simulated inter-datacenter network is in consistent with our real-world experiment setting, where 6 geographically distributed datacenters are interconnected by links with bandwidth capacities shown in Table 5.1. For simplicity, each job has 10 tasks and each task has three partitions of input data, which are randomly distributed across all the datacenters. The size of input data is uniformly distributed in the range of $[50, 600]$ MB. The total resource capacity, *i.e.*, the total amount of available computing slots, across all the datacenters is set as $1, 1.1, 1.5, 2.5, 5$ and 10 times the total number of tasks (represented as $1X$, $1.1X$, $1.5X$, $2.5X$, $5X$ and $10X$), respectively in each group of simulation, to represent the different degrees of resource competition.



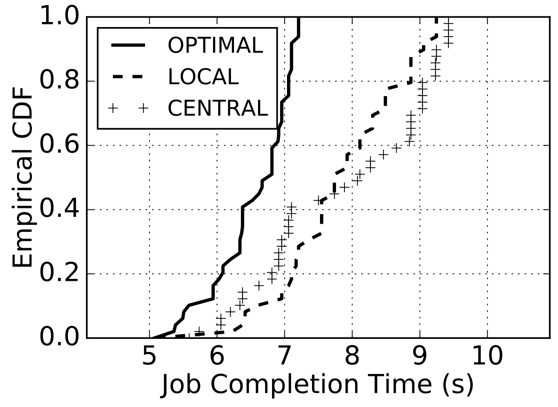
(a) Even and Loose ($2X$)



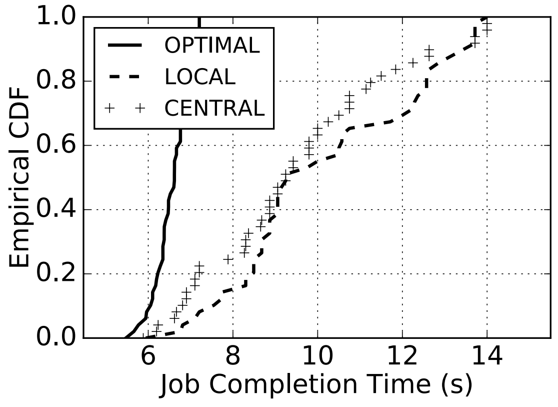
(b) Even and Tight ($1.1X$)

Figure 5.12: The CDFs of job completion times for 50 concurrent jobs, achieved with three algorithms, with different resource availability.

We compare our proposed algorithm, referred to as *OPTIMAL*, with two baselines — *LOCAL* and *CENTRAL*. *LOCAL* refers to the default Spark scheduling algorithm applied in the wide-area scenario, which tries to allocate tasks to the datacenter where their input data are stored. If such a data locality can not be satisfied for the lack of sufficient resources, *LOCAL* will randomly select a datacenter with available resources for task assignment. *CENTRAL* refers to the algorithm which attempts to assign all the tasks of a job to one or a few main datacenters as long as computing slots are available. It represents the most naive way of running data analytics in the wide area,

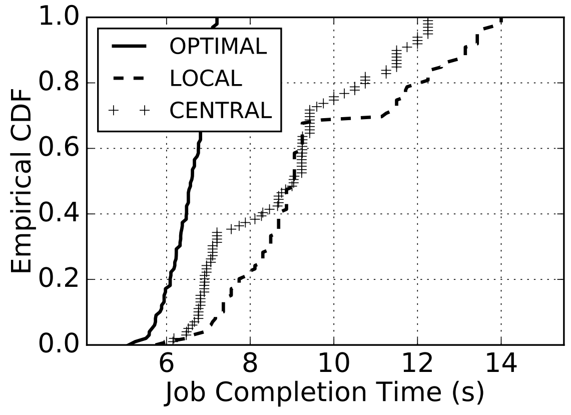


(a) Skew and Loose ($2X$)

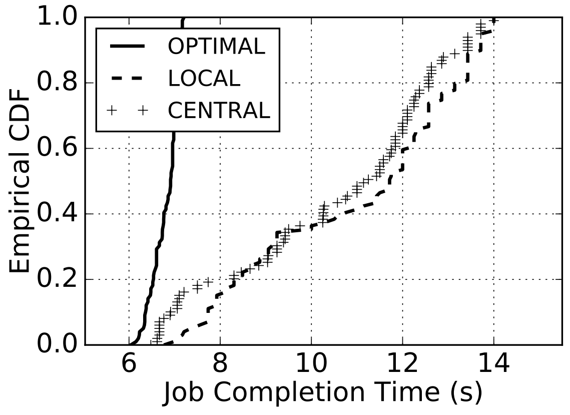


(b) Skew and Tight ($1.1X$)

Figure 5.13: The CDFs of job completion times for 50 concurrent jobs, achieved with three algorithms, with different resource availability.



(a) Even and Loose ($2X$)



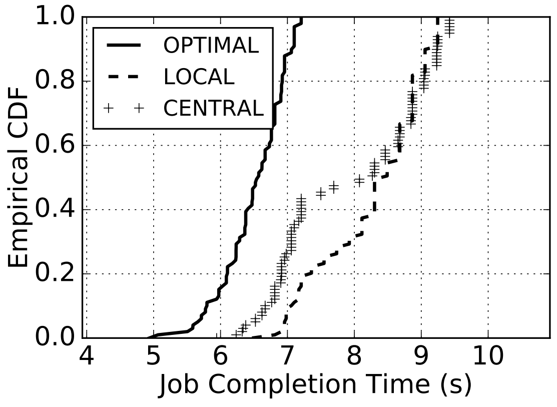
(b) Even and Tight ($1.1X$)

Figure 5.14: The CDFs of job completion times for 100 concurrent jobs, achieved with three algorithms, with different resource availability.

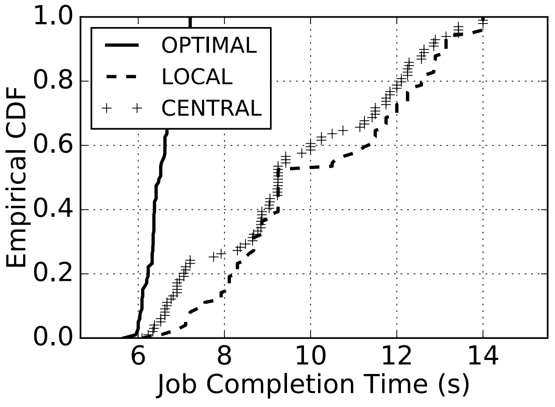
which aggregates all the input data to a central datacenter and thus run all the tasks within a single cluster.

Our primary evaluation metrics are the completion times of concurrent jobs and the improvement ratio of the worst performance among concurrent jobs. To be more specific, for a job whose completion time is *opt* with our algorithm and *baseline* with a baseline algorithm, the improvement ratio is calculated as $(opt - baseline)/baseline \times 100\%$, of which the maximum is 100%.

**Impact of Resource Availability.** We first present a group of results with different



(a) Skew and Loose ($2X$)



(b) Skew and Tight ($1.1X$)

Figure 5.15: The CDFs of job completion times for 100 concurrent jobs, achieved with three algorithms, with different resource availability.

settings of resource availability. Fig. 5.12, Fig. 5.13, Fig. 5.14 and Fig. 5.15 illustrate the empirical CDFs of job completion times across all the 50 and 100 concurrent jobs, respectively. Each subfigure presents the CDFs of job completion times achieved with the three comparing algorithms (*OPTIMAL*, *LOCAL* and *CENTRAL*), given different settings of resource distribution and competition degree. It is easily observed in all the figures that our algorithm always outperforms the baselines with respect to improving the worst job completion time.

*Competition Degree: Tight vs. Loose.* Fig. 5.12a and Fig. 5.12b illustrate the completion times of 50 concurrent jobs when all the available computing slots are evenly distributed across all the datacenters. The degrees of resource competition are set as $2X$ (*loose*) and $1.1X$ (*tight*) for Fig. 5.12a and Fig. 5.12b, representing that the total numbers of available slots are 2 and 1.1 times the total number of tasks, respectively. As we observe, when the resource competition becomes more intense, *i.e.*, there is a tight budget of resource, the difference between our algorithm and the baselines becomes more obvious. This implies that our algorithm shows more benefits in resolving competitions with our max-min fairness when the budget of resource is tight. In a similar vein, when the available computing slots are skew in distribution, our *OPTIMAL* also achieves

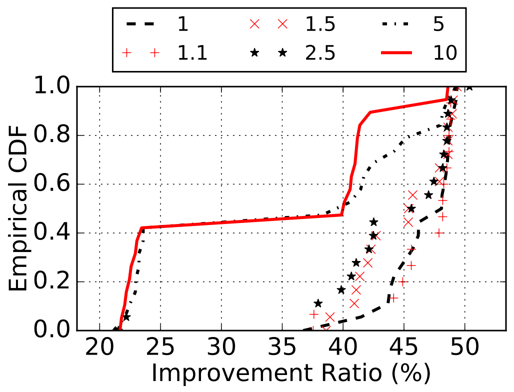


Figure 5.16: CDFs of improvement ratios for 50 jobs over 20 runs. (*OPTIMAL* vs. *LOCAL*)

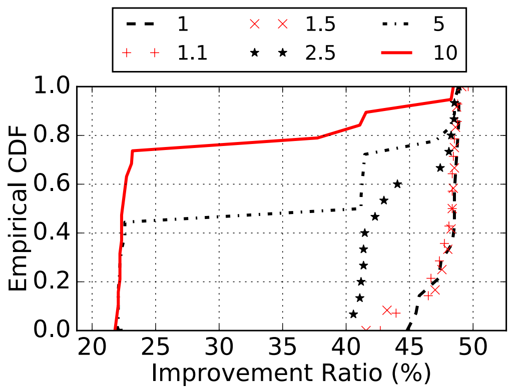


Figure 5.17: CDFs of improvement ratios for 100 jobs over 20 runs. (*OPTIMAL* vs. *LOCAL*)

shorter job completion times than the baselines, and the advantage has an increasing trend with a tighter budget of resources, as demonstrated by Fig. 5.13a and Fig. 5.13b. The similar analysis applies to Fig. 5.14 and Fig. 5.15 when the number of concurrent jobs is increased to 100.

*Resource Distribution: Skew vs. Even.* Given a fixed degree of resource competition, our algorithm shows more advantage in the case of a even resource distribution, compared with a skew one. This can be easily demonstrated by comparing Fig. 5.12a and Fig. 5.13a for the *loose* case, and comparing Fig. 5.12b and Fig. 5.13b for the *tight* case. Also, Fig. 5.14 and Fig. 5.15 show a similar pattern. The explanation for such a pattern is that when the resource is evenly distributed, our algorithm has a larger space for optimizing the task placement, and thus exhibit more performance improvement.

**Average Improvement Ratio.** Apart from the detailed case study for Fig. 5.12, Fig. 5.13, Fig. 5.14 and Fig. 5.15, we further present performance improvement ratios achieved with 6 degrees — $1X$, $1.1X$, $1.5X$, $2.5X$, $5X$ and $10X$ — of resource competition, for 50 and 100 jobs, respectively, where the available computing slots are randomly distributed across the datacenters. Specifically, the performance improvement ratio is the percentage of reduction in the worst completion time among all the concurrent jobs achieved by our *OPTIMAL*, compared with the baseline *LOCAL*. Fig. 5.16 and Fig. 5.17

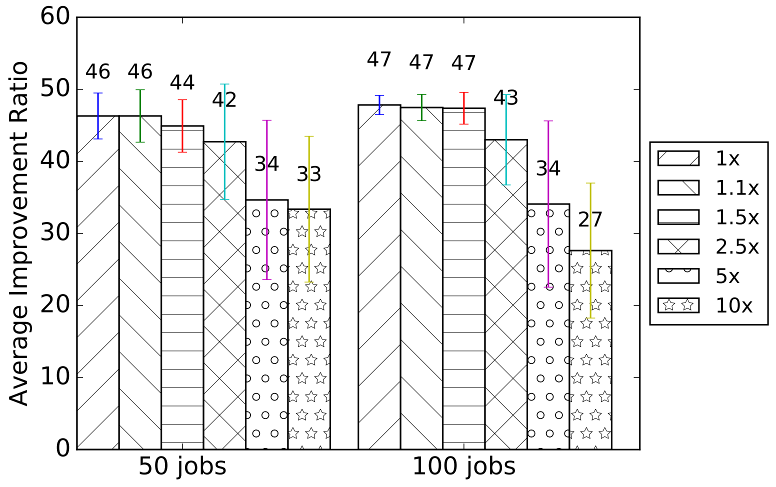
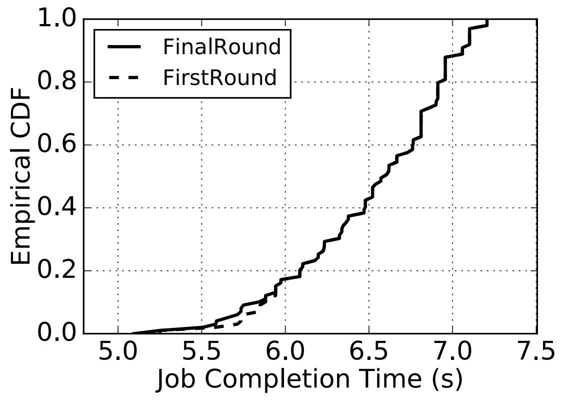


Figure 5.18: Average improvement ratios with different resource availability for 50 and 100 jobs.

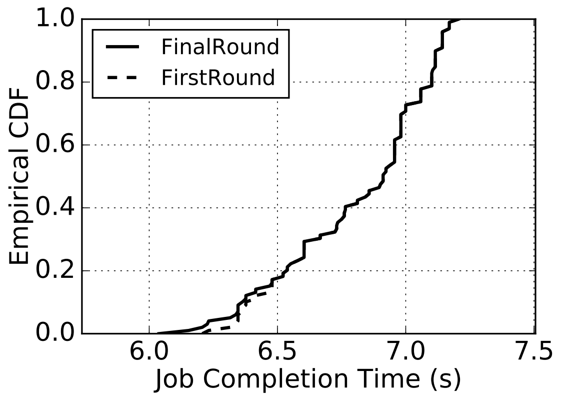
show the empirical CDFs of the improvement ratio over 20 runs, for 50 and 100 concurrent jobs, respectively. Moreover, the average performance improvement ratios are presented in Fig. 5.18 for intuitive comparison.

Clearly, our algorithm achieves a performance improvement of 27% to 47% over *Local*. With a larger degree of resource competition (*i.e.*, with more available computing slots), the improvement ratio becomes smaller. The reason is that our algorithm always seeks for the optimal solution in minimizing the worst job completion time, despite the degree of competition. However, the performance of $LOCAL$ is impacted by the resource availability. With more resources, $LOCAL$ can achieve data locality for more tasks, which avoids stragglers and improves job completion times. Therefore, the advantage of our algorithm becomes less obvious for the $2.5X$, $5X$ and $10X$ cases. The higher standard deviation for these cases can be explained by the randomness in placing tasks that can not achieve data locality in $LOCAL$. Though the total amount of slots is larger, some tasks may still fail to achieve data locality, as the resources are skew in distribution.

**Improvement Over Rounds with *OPTIMAL*.** Now we focus on the behavior of our algorithm by investigating the calculated task scheduling decisions over rounds. To be particular, we calculate the job completion times if scheduled with the temporary decision

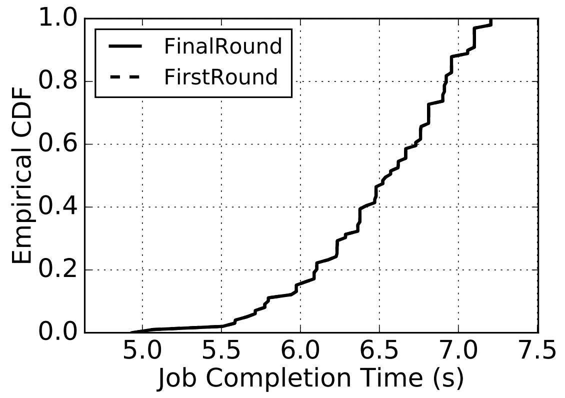


(a) Even and Loose ($2X$)

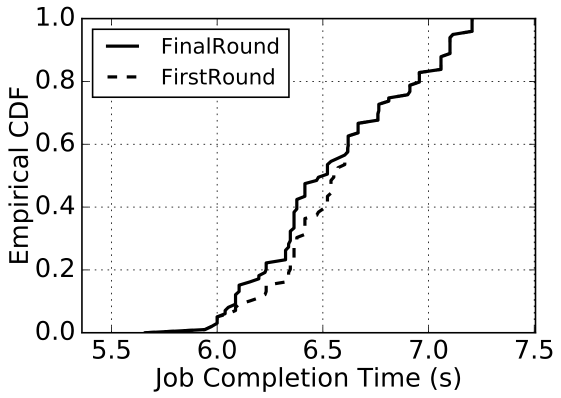


(b) Even and Tight ($1.1X$)

Figure 5.19: The CDFs of job completion times for 100 concurrent jobs, achieved in the first round and the final round of *OPTIMAL*.



(a) Skew and Loose ($2X$)



(b) Skew and Tight ($1.1X$)

Figure 5.20: The CDFs of job completion times for 100 concurrent jobs, achieved in the first round and the final round of *OPTIMAL*.

calculated in the first round and the final decision after the final round, respectively. We find that the job completion times achieved in these two rounds are the same for most cases. For the remaining cases, the improvement achieved in the final round over the first round is marginal. We present the CDFs of completion times for 100 concurrent jobs achieved in the first round and the final round of our algorithm, respectively, in Fig. 5.19 and Fig. 5.20. The settings of the four subfigures are the same with Fig. 5.14 and Fig. 5.15. As clearly illustrated, the performance in the final round shows no difference with the first found in Fig. 5.20a, and is slightly better in Fig. 5.19a, 5.19b and 5.20b.

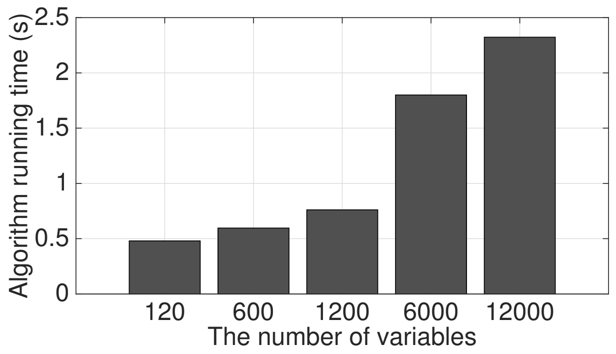


Figure 5.21: The computation times of our algorithm in large-scale simulations, with different scales.

The improvement is due to the iterative optimization of our algorithm in improving performance of some jobs without impacting others, which has been clearly illustrated in Sec. 5.4. These figures imply that the optimization in the first round is the major contributor to the performance improvement of our algorithm. The space for further improvement over later rounds depends on specific cases and is not large.

**Scalability.** Furthermore, as in our real-world experiment, we have measured the running times with different problem scales in our simulations to evaluate the scalability of our algorithm. The measured computation time with each problem scale is averaged over 10 runs. Fig. 5.21 illustrates the computation times when the number of variables ranges from 120 to 12000. As the `CPLEX` solver used in simulations is more efficient, solving the problem with 120 variables only takes less than 0.5 seconds, which is much faster than the solver in `Breeze` used in our experiment. When the number of variables grows beyond ten thousands, the computation time is still less than 2.5 seconds.

## 5.6 Discussion

**Adaptation to Network Heterogeneity.** Distributed data analytics depends heavily on the reliability and the performance of the underlying networks, due mainly to its abundant demand for data exchange among tasks. However, the networks, being best-effort and shared among many, can hardly deliver consistent or predictable Quality of

Service, especially in the wide area networks. As a result, data analytics jobs usually suffer from performance degradation due to *spatial* and *temporal* network heterogeneity: different links can offer different capacities at the same time, while the available capacity of a link can vary over time. It remains an open problem to deal with these two types of heterogeneity in data analytics systems adaptively and in real-time. Solving this problem requires to identify the heterogeneity effectively and to adjust the job scheduling with the awareness of application-level workload. Our work offers a preliminary solution which accounts for the spatial network heterogeneity. To be adaptive to the temporal heterogeneity, we can monitor the bandwidth and run our algorithm periodically or when the bandwidth dynamics exceeds a certain threshold.

**Economic-Based Resource Allocation.** It would be interesting to investigate the problem of task assignment for data analytic jobs among geo-distributed datacenters from the game theoretic perspective. For example, in the framework of auctions, each job can be viewed as a buyer bidding for multiple goods, in the form of computing slots, from multiple datacenters. Different from conventional resource allocation where the utility of a user is determined by the summation of the total amount of resources received, allocating resources for data analytic jobs suffers from the challenge that the job performance is determined by the slowest task, which means that the utility is not determined by a simple summation of resources. Such a non-additive nature may significantly change the problem structure and introduce complexities in identifying the optimal solution. It is an interesting direction to be explored in our future work.

## 5.7 Summary

In this chapter, we have conducted a theoretical study of the task assignment problem among competing data analytic jobs, whose input data are distributed across geo-distributed datacenters. With tasks from multiple jobs competing for the computing slots in each datacenter, we have designed and implemented a new optimal scheduler to assign tasks across these datacenters, in order to better satisfy job requirements with max-min fairness achieved across their job completion times. To achieve this objective, we first formulated a lexicographical minimization problem to optimize all the job completion times, which is challenging due to the inherent complexity of both multi-objective and discrete optimizations. To address these challenges, we started from the single-objective subproblem and transformed it into an equivalent linear programming (LP) problem to be efficiently solved in practice, based on an in-depth investigation of the problem structure. An algorithm is further designed to repeatedly solve an updated version of the LP subproblems, which would eventually optimize all the job performance with max-min fairness achieved. Last but not the least, we have implemented our performance-optimal scheduler in the popular Spark framework, and demonstrated convincing evidence on the effectiveness of our new algorithm using both real-world experiments and large-scale simulations.

# Chapter 6

# Multi-path Routing and Scheduling across Datacenters

To optimize the performance of data analytic jobs deployed in the wide area, we further exploit the flexibility of multi-path routing and scheduling in this chapter. Existing efforts either reduce the total volume of inter-datacenter traffic or design better task placement for load balancing, which all require modifying the generated traffic pattern across datacenters to alleviate the performance degradation of inter-datacenter transfers. Complementary and orthogonal to these work, we focus on directly optimizing the network transfers given certain traffic patterns. Particularly, we propose to optimize the inter-datacenter transfers by exploiting the path flexibility to better utilize the inter-datacenter link bandwidth and thus accelerate the geo-distributed jobs eventually.

Our strategy design is based on a rigorous formulation of bandwidth allocation and multi-path routing problem among sharing flows across datacenters. With a theoretical study of the problem, we design an optimal strategy and implemented it in the central controller in the application-layer software-defined network. With extensive real-world experiments, our strategy is demonstrated to achieve optimal job performance, with max-min fairness achieved among sharing jobs deployed in the wide area.

## 6.1 Background and Motivation

In this section, we first present an overview of the execution of a data analytics job whose input data is stored across geographically distributed datacenters. Then, motivated by the flexibility of available paths for network flows across these datacenters, we will illustrate the intuitive and immediate benefit of utilizing multiple paths in transferring a coflow.

**Geo-Distributed Data Analytics.** Data parallel frameworks, such as MapReduce and Spark, have become the mainstream platform for today's large-scale data analytics. A typical data analytics job using these frameworks proceeds through several computation stages, each consisting of tens or hundreds of parallel computation tasks. Between consecutive stages, network flows will be generated in the communication stage to transfer the intermediate data for further processing.

When input data of such data analytics jobs are globally generated and stored across datacenters in different regions, which becomes increasingly typical for global services provided by today's big companies [26, 12, 13], new challenges arise in running analytics jobs to process such geo-distributed data efficiently. Particularly, there are two alternatives for such data processing. A naive approach is to gather all the input data to a single datacenter and run the traditional data analytics job, which involves an extensive amount of data to be transferred across datacenters. A more advanced alternative is to distribute tasks across datacenters without centralizing the input data, which incurs a smaller amount of intermediate data to be exchanged among datacenters.

In contrast with the datacenter network, the inter-datacenter links have much smaller bandwidth and are heterogeneous with respect to both the available bandwidth and the network loads [70], which make the communication stage easily become the performance bottleneck. Since cross-datacenter transfers are inevitable in geo-distributed data analytics, it becomes crucial to optimize the cross-datacenter network transfers to accelerate job execution and improve job performance.

**Flexibility in Path Selection.** Inter-datacenter data transfers, in essence, are wide-area network transfers, usually through dedicated links. Google [37] reported that dedicated optical links have been deployed to inter-connect their globally distributed datacenters. As a result, there exist multiple available paths between any pair of datacenters and, more importantly, they are accessible and controllable by users. For example, to transfer bulk data, one can use a direct transfer between the source and the destination datacenters or, in case of congestion on the direct link, explicitly utilize an intermediate datacenter as a relay to achieve a higher throughput.

In our context of running data analytics jobs that is deployed across different datacenters, this property can be helpful in terms of reducing the coflow completion times. Fig. 1.1 shows a toy example, when a large amount of intermediate data need to be transferred from N. Virginia to Singapore between mappers and reducers. Apparently, the narrow direct link will become the bottleneck, slowing down the completion of the communication stage.

To alleviate the bottleneck, we can route some traffic through alternative paths with larger available bandwidth, such as the detoured path through Oregon or N. California as aforementioned. Intuitively, if all the flows generated in the communication stage can be better balanced across the inter-datacenter links, the slowest flow will finish faster, which will result in faster completion of the communication stage.

Therefore, our design objective in this chapter is to exploit such flexibilities in path selection to optimally coordinate the inter-datacenter network flows of data analytics jobs, so that their communication stages would complete faster and thus the jobs would be accelerated. We would also account for the fairness issue when network flows from multiple jobs are concurrently sharing the inter-datacenter network.

**Optimal Multi-Path Routing.** We next use a more detailed example to illustrate the basic idea of our multi-path routing in accelerating a data analytics job. As shown in Fig. 6.1, we consider a MapReduce job which has a mapper task `M1` in `DC1` (Datacenter

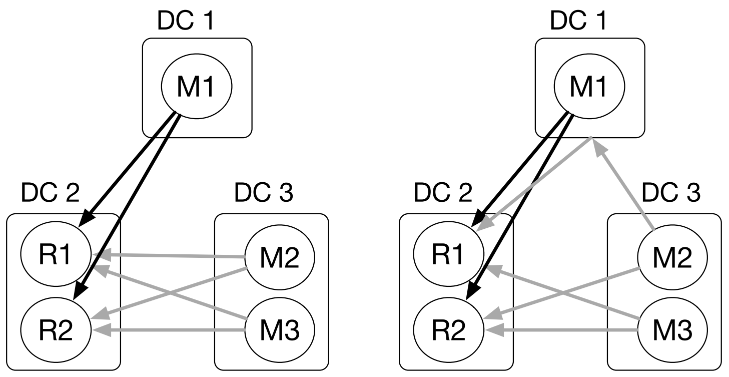


Figure 6.1: Routing with direct paths vs. Optimal multi-path routing.

1), two mapper tasks `M2` and `M3` in `DC3` and other mapper tasks (omitted in the figure) in `DC2`. The reduce tasks `R1` and `R2` are launched in `DC2`, which need to fetch intermediate data from mappers during the shuffle stage. In particular, four flows will be generated from `DC3` to `DC2`, represented by the gray lines, and two flows from `DC1` to `DC2` which are represented by the black lines. (We only illustrate the cross-datacenter flows in the figure.) For simplicity, each of these flows is 100 MB in size, and the link bandwidth between each pair of datacenters is the same (10 MB/s).

If all these flows are routed through their direct paths as on the left side of Fig. 1.1, the link between `DC3` and `DC2` would become the bottleneck, resulting in 40 seconds of shuffle completion time (each of the four flows gets a fair share of 2.5 MB/s). However, if we split some traffic from the direct path (`DC3-DC2`) to the alternative path (`DC3-DC1`, `DC1-DC2`), the network load will be better balanced across inter-datacenter links, which naturally speeds up the shuffle phase. Specifically, when 100 MB of traffic is shifted from `DC3-DC2` to the alternative two-hop path, as illustrated by the right side of Fig. 1.1, both `DC3-DC2` and `DC1-DC2` have the same network load. With this routing, the shuffle completion time is reduced to only 30 seconds (on both `DC3-DC2` and `DC1-DC2` links, each of the three sharing flows gets a fair share of 10/3 MB/s).

Apart from the flexibility in multi-path routing, when we further consider the inter-datacenter flows from multiple jobs and the flexibility of rate assignment, a larger space can be explored to optimize the network transfers and eventually the job performance for all the sharing jobs, with fairness constraints. Such an extension beyond the simple example would be elaborated with formal formulations and detailed analysis in our later sections.

## 6.2 Optimal Transfer for A Single Job

In this section, as a starting point, we will formally construct a mathematical model to study the general problem of multi-path routing for cross-datacenter traffic generated by a single data analytic job, with the objective of minimizing the completion time of the communication stage.

For a typical MapReduce job, the communication stage is called a *shuffle*, and the set of all the network flows in a shuffle is defined as a *coflow* [7]. In the multi-datacenter scenario, the group of flows in a coflow that have the same pair of source and destination datacenters is called an *aggregated flow*, which is treated as the basic unit in our multi-path routing problem for convenience.

In our consideration, a job generates $I$ aggregated flows in its shuffle. The volume of traffic for the $i$-th aggregated flow is denoted by $d_i$, which is the total amount of data to be sent by all the consisting flows. Each aggregated flow has the flexibility to take either the direct path, through the link between the source and destination datacenters; or take the detoured path, using other datacenters as relays. The number of total available paths for the $i$-th aggregated flow is denoted as $J_i$, among which the $j$-th ($j \leq J_i$) path is denoted as $p_{i,j}$, the set of inter-datacenter links along the path.

Since different links may have different amounts of traffic, with the flexibility of multi-path routing, the network load can be better balanced across all the links, so that the whole shuffle stage can complete faster. Therefore, we would like to design the optimal multi-path routing algorithm that minimizes the shuffle completion time, *i.e.*, the completion time of the slowest aggregated flow. The decision variables in our routing problem are denoted by $\alpha_{i,j}, \forall i \in \{1, 2, \cdots, I\}, \forall j \in \{1, 2, \cdots, J_i\}$, representing the percentage of traffic of the $i$-th aggregated flow that is routed through its $j$-th path.

Let $\mathcal{L} = \cup_{i,j} p(i,j), \forall i, j \in \{1, 2, \cdots, J_i\}$ denote the set of all the inter-datacenter links along paths of all the aggregated flows in our consideration. For each link $l \in \mathcal{L}$, the available link bandwidth is represented by $b_l$. If $l$ is along the $j$-th path of the $i$-th aggregated flow, *i.e.*, $l \in p(i,j)$, $\alpha_{ij}$ percentage of the $i$-th aggregated flow will pass through this link, which increases the link load by the amount of $d_i\alpha_{ij}$. Since $l$ may belong to multiple paths of a particular aggregated flow, and may also belong to paths of different aggregated flows, the total amount of traversing traffic is calculated as the summation of the traffic load over all the possible paths $j$ ($j \in \{1, 2, \cdots, J_i\}$) of a particular aggregated flow $i$ it belongs to ($l \in p(i,j)$), and further over all the considered aggregated flows $i \in \{1, 2, \cdots, I\}$, which is eventually represented as $\sum_i \sum_{j, l\in p(i,j)} d_i\alpha_{ij}$. Therefore, the time it takes to complete all the traffic routed through link $l$ is calculated as $t_l = \sum_i \sum_{j, l\in p(i,j)} d_i\alpha_{ij}/b_l$.

Since the shuffle completion time is the slowest completion time among all the aggregated flows, and the completion time of an aggregated flow is determined by the bottleneck link along all the paths, we can easily represent the shuffle completion time as $t = \max_{l\in\mathcal{L}} t_l$.

Substituting the expression of $t_l$ in $t$, we obtain our optimal multi-path routing problem, which is aimed at minimizing the shuffle completion time, as follows:

$$\min_{\alpha_{ij}} \quad t \tag{6.1}$$

$$\text{s.t.} \quad t = \max_{l\in\mathcal{L}} \frac{\sum_i \sum_{j, l\in p(i,j)} d_i\alpha_{ij}}{b_l} \tag{6.2}$$

$$\sum_{j=1}^{J_i} \alpha_{ij} = 1, \forall i \in \{1, 2, \cdots, I\} \tag{6.3}$$

$$\alpha_{ij} \geq 0, \forall i, j \in \{1, 2, \cdots, J_i\} \tag{6.4}$$

Constraint (6.3) indicates that for each aggregated flow, the sum of traffic percentages allocated to each of its paths is 1, representing the nature of ratios. Constraint (6.4) also stands for the positive nature of ratios.

This problem is equivalent to the following problem:

$$\begin{aligned} \min_{\alpha_{ij},\lambda} \quad & \lambda && (6.5) \\ \text{s.t.} \quad & \lambda \geq \frac{\sum_i \sum_{j,l\in p(i,j)} d_i \alpha_{ij}}{b_l}, \forall l \in \mathcal{L} && (6.6) \\ & \text{Constraints (6.3) and (6.4).} \end{aligned}$$

The intuition is that for any feasible $\lambda$, we have $\lambda \geq t = \max_{l\in\mathcal{L}} \frac{\sum_i \sum_{j,l\in p(i,j)} d_i \alpha_{ij}}{b_l}$. It is obvious that the objective achieves the minimum only when $\lambda = t$. Therefore, Problem (6.1) and Problem (6.5) are equivalent.

Further, we transform constraint (6.6) to an equivalent form, which gives the equivalent optimization problem as follows:

$$\begin{aligned} \min_{\alpha_{ij},\lambda} \quad & \lambda && (6.7) \\ \text{s.t.} \quad & \sum_i \sum_{j,l\in p(i,j)} d_i \alpha_{ij} \leq \lambda b_l, \forall l \in \mathcal{L} && (6.8) \\ & \text{Constraints (6.3) and (6.4).} \end{aligned}$$

As observed, the objective is linear, and all the constraints are linear as well. Hence, Problem (6.7) is a linear programming (LP) problem, which can be solved using efficient LP solvers, such as MOSEK [17].

**Selecting the best one**. It is worth noting that there may exist multiple optimal solutions to Problem (6.1), which result in the same shuffle completion time.

Among these equivalent solutions, we would like to select the one that incurs the minimum overhead with respect to traffic shifting. The principle is that direct paths should be used with priority. Following this principle, we add a penalty function $g(\alpha_{i,j})$

for traffic shifting in the objective:

$$g(\alpha_{i,j}) = -\delta \sum_{i,|p(i,j)|=1} \alpha_{ij}, \tag{6.9}$$

where $|p(i,j)| = 1$ indicates that the $j$-th path of the $i$-th aggregated flow is a direct path, which consists of only one direct link. The expression $\sum_{i,|p(i,j)|=1} \alpha_{ij}$ represents the sum of direct path traffic percentages over all the transfers. $\delta$ is a positive parameter which is tuned so that $|g(\alpha_{i,j})|$ is at least two orders of magnitude smaller than $\lambda$.

With the penalty function added, we obtain the following optimization problem:

$$\begin{aligned} \min_{\alpha_{ij},\lambda} \quad & \lambda - \delta \sum_{i,|p(i,j)|=1} \alpha_{ij} && (6.10) \\ \text{s.t.} \quad & \text{Constraints (6.8), (6.3) and (6.4).} && (6.11) \end{aligned}$$

The more direct paths used, the smaller the penalty function will be, and thus the smaller the objective. Since the problem is a minimization problem, the solution that incurs the least amount of traffic to be shifted away from their direct paths will be selected, as we desire.

It is obvious that the objective is still linear. Hence, Problem (6.10) is an LP that can be efficiently solved.

## 6.3 Multi-Path Routing and Scheduling for Sharing Jobs

We now consider a general scenario where $I$ aggregated flows from multiple concurrent data analytic jobs are sharing the inter-datacenter network.

If our objective is to minimize the slowest shuffle completion time among all the jobs, then the problem formulation in the previous section is readily applicable to this multi-job scenario. The only difference is that the $I$ aggregated flows are generated from multiple concurrent jobs rather than a single job in the previous section.

However, when we try to further improve the completion time for the next slowest shuffle, the previous formulation is no longer applicable, because of the limit of the default TCP fair sharing it relies upon. In this multi-job sharing scenario, rate assignment plays a significant role in improving the effective utilization. Hence, we modify our previous problem formulation, to involve the flexibility of rate assignment (or *scheduling*), so that an even more flexible solution can be devised to improve shuffle completion times.

### 6.3.1 Optimizing the Worst

We first consider the formulation with the objective of accelerating the slowest shuffle, , *i.e,*, minimizing the maximal shuffle completion time among all the sharing jobs.

For the aggregated flow $i \in \mathcal{I} = \{1, 2, \cdots, I\}$, the set of all its available paths is denoted as $\mathcal{P}_i$. We allow such a flow to be flexibly split to take any of its paths, at specific sending rates. To be more specific, we use $x_i(p), \forall i \in \mathcal{I}, \forall p \in \mathcal{P}_i$ to represent the sending rate or throughput of the $i$-th aggregated flow along its path $p$, which is the decision variable in our multi-path routing and scheduling problem. Intuitively, the total throughput achieved along all the paths of flow $i$ is represented as $\sum_{p \in \mathcal{P}_i} x_i(p)$. Hence, its completion time is derived as $d_i / \sum_{p \in \mathcal{P}_i} x_i(p)$, representing the flow size divided by the total throughput.

On each link $l \in \mathcal{L}$, the total rate of all the traversing flows should not exceed its available bandwidth capacity $b_l$. Let $\lambda_i(p, l) \in \{0, 1\}$ represent whether link $l$ is along the path $p$ of the $i$-th aggregated flow. The total throughput of all the traffic along the link $l$ is calculated as $\sum_{i \in \mathcal{I}} \sum_{p \in \mathcal{P}_i} x_i(p)\lambda_i(p, l)$, which is the summation of the throughput over all the possible paths of an aggregated flow that link $l$ belongs to, and the summation over all the aggregated flows. Hence, we obtain the link capacity constraints as follows:

$$\sum_{i \in \mathcal{I}} \sum_{p \in \mathcal{P}_i} x_i(p)\lambda_i(p, l) \le b_l, \quad \forall l \in \mathcal{L}$$

If we use $T$ to represent the completion time of the slowest shuffle, we have the following optimization problem:

$$\min_{x_i(p)} \quad T \tag{6.12}$$

$$\text{s.t.} \quad T = \max_{i\in\mathcal{I}} d_i / \sum_{p\in\mathcal{P}_i} x_i(p) \tag{6.13}$$

$$\sum_{i\in\mathcal{I}} \sum_{p\in\mathcal{P}_i} x_i(p)\lambda_i(p,l) \le b_l, \quad \forall l \in \mathcal{L} \tag{6.14}$$

$$x_i(p) \ge 0, \quad \forall p \in \mathcal{P}_i, \quad i \in \mathcal{I} \tag{6.15}$$

Constraint (6.13) indicates that $T$ is the maximum completion time among all the shuffles. Constraint (6.14) is the bandwidth capacity constraint as aforementioned.

With the same justification as in the previous section, we transform Problem (6.12) to the following equivalent problem:

$$\min_{x_i(p),\gamma} \quad \gamma \tag{6.16}$$

$$\text{s.t.} \quad \gamma \ge d_i / \sum_{p\in\mathcal{P}_i} x_i(p), \qquad \forall i \in \mathcal{I} \tag{6.17}$$

Constraints (6.14) and (6.15).

If we denote $\mathcal{X} = 1/\gamma$, then by substituting $\gamma = 1/\mathcal{X}$, Problem (6.16) is rewritten as the following form:

$$\min_{x_i(p),\mathcal{X}} \quad 1/\mathcal{X} \tag{6.18}$$

$$\text{s.t.} \quad 1/\mathcal{X} \ge d_i / \sum_{p\in\mathcal{P}_i} x_i(p), \qquad \forall i \in \mathcal{I}$$

Constraints (6.14) and (6.15).

It is obvious that $\mathcal{X} = 1/\gamma > 0$, thus we can further transform our original problem to the equivalent problem as follows:

$$\max_{x_i(p),\mathcal{X}} \quad \mathcal{X} \tag{6.19}$$

$$\text{s.t.} \quad 0 < \mathcal{X} \leq \sum_{p \in \mathcal{P}_i} x_i(p)/d_i, \qquad \forall i \in \mathcal{I} \tag{6.20}$$

$$\text{Constraints (6.14) and (6.15).}$$

It is easy to check that the objective and constraints are both linear. Therefore, our multi-path routing and scheduling problem is equivalent to a linear programming problem. This problem only has $\sum_i |\mathcal{P}_i| + 1$ variables and $\sum_i |\mathcal{P}_i| + I + |\mathcal{L}| + 1$ constraints, where $|\mathcal{P}_i|$ represents the number of paths for aggregated flow $i$ and $|\mathcal{L}|$ denotes the number of inter-datacenter links. Given the limited number of datacenters in practice, both $|\mathcal{P}_i|$ and $|\mathcal{L}|$ are very small. Hence, this LP problem is of small scale and can be efficiently solved.

### 6.3.2 Continuously Optimizing the Next Worst

With our primal objective of minimizing the worst completion time efficiently solved as an LP problem (6.19), we continue to improve the bandwidth utilization to minimize the next worst completion time repeatedly.

The improvement of bandwidth utilization relies on the awareness of the coflow semantics. If two aggregated flows belong to the same job, which means that they come from the same coflow, then the coflow completion time is depending on the slowest one.

Let $\mathcal{C}_i$ represent the coflow that flow $i$ belongs to. $\mathcal{C}_i$ is the set of all its constituting flows. If flow 1 and 2 belong to the same coflow, then we have $\mathcal{C}_1 = \mathcal{C}_2 \supseteq \{1, 2\}$.

After we optimize the worst completion time ($1/\mathcal{X}^*$), we fix the rate assignment for the flow $i^*$ that achieves this completion time. Then, for all the other flows that belong to the same coflow $\mathcal{C}_{i^*}$, they do not need to finish faster than flow $i^*$, as it does not help improve the coflow completion time. Therefore, we can let them finish at the same time with the slowest flow $i^*$, by adding the following equality constraint:

$$\sum_{p \in \mathcal{P}_i} x_i(p)/d_i = \mathcal{X}^*, \qquad \forall i \in \mathcal{C}_{i^*}$$

Then in the next round, we try to optimize the completion time of the next worst flow, which belongs to a different coflow and determines the completion time of the coflow:

$$\begin{aligned} \max_{x_i(p), \mathcal{X}} \quad & \mathcal{X} \\ \text{s.t.} \quad & 0 < \mathcal{X} \le \sum_{p \in \mathcal{P}_i} x_i(p)/d_i, \qquad \forall i \in \mathcal{I}' \\ & \sum_{i \in \mathcal{I} - \tilde{\mathcal{I}}} \sum_{p \in \mathcal{P}_i} x_i(p)\lambda_i(p,l) \le b'_l, \quad \forall l \in \mathcal{L} \\ & x_i(p) \ge 0, \quad \forall p \in \mathcal{P}_i, \quad i \in \mathcal{I} - \tilde{\mathcal{I}} \\ & \sum_{p \in \mathcal{P}_i} x_i(p)/d_i = \mathcal{X}^*, \qquad \forall i \in \mathcal{C}_{i^*} \end{aligned} \tag{6.21}$$

where $\mathcal{I}'$ represents the set of flows whose coflow does not have any flow assigned with rate, and $\tilde{\mathcal{I}}$ denotes the set of all the flows whose rates have been assigned in previous rounds. $b'_l = b_l - \sum_{i \in \tilde{\mathcal{I}}} \sum_{p \in \mathcal{P}_i} x_i(p)\lambda_i(p,l)$, which is the updated available bandwidth.

In this way, we avoid improving the completion time of the flow that belongs to the same coflow with previously allocated flows. This guarantees that the bandwidth is efficiently used to improve the completion time of a new coflow round by round, rather than improving the flow whose coflow completion time is already determined in previous rounds.

As a result, the optimization problem in the next round is solved over a decreased set of variables with updated constraints and objectives, so that the next worst coflow completion time would be optimized, without impacting the coflow completion time optimized in this round. Such a procedure is repeatedly executed until the completion time of the last coflow has been optimized, as summarized in Algorithm 5. Eventually,

**Algorithm 5:** Performance-Optimal Multi-Path Rate Assignment for Inter-Datacenter Flows from Sharing Jobs with Max-Min Fairness.

```
Input:
    The aggregated flow set $\mathcal{I}$; the coflow set $\mathcal{C}_i, \forall i \in \mathcal{I}$;
    The total amount of data to be sent by each aggregated flow $i$: $d_i$;
    The set of available paths for each aggregated flow $i$: $\mathcal{P}_i$;
Output:
    Rate assignment for each aggregated flow along each path: $x_i(p), \forall i \in \mathcal{I}, \forall p \in \mathcal{P}_i$;
1: Initialize $\mathcal{I}' = \mathcal{I}, \tilde{\mathcal{I}} = \varnothing$;
2: while $\mathcal{I}' \neq \varnothing$ do
3:     Solve the LP Problem (6.19) to obtain the solution $\boldsymbol{x^*}, \mathcal{X}^*$;
4:     Obtain $x_{i^*}(p), \forall p \in \mathcal{P}_{i^*}$, which satisfies $\sum_{p \in \mathcal{P}_{i^*}} x_{i^*}(p)/d_{i^*} = \mathcal{X}^*$;
5:     Fix $x_{i^*}(p), \forall p \in \mathcal{P}_{i^*}$; remove them from the variable set (by adding $i^*$ to $\tilde{\mathcal{I}}$);
6:     Update the corresponding link bandwidth capacities in Constraints (6.14);
7:     Find other aggregated flows that belong to the same coflow with $i^*$, add equality
       constraints: $\sum_{p \in \mathcal{P}_i} x_i(p)/d_i = \mathcal{X}^*,\ \forall i \in \mathcal{C}_{i^*}$;
8:     Remove $\mathcal{C}_{i^*}$ from $\mathcal{I}'$;
9: end while
```

all the coflows achieve their best possible completion times, and max-min fairness can be achieved.

## 6.4 Implementation

We have implemented our multi-path routing and scheduling strategy in our software-defined inter-datacenter overlay testbed, which provides a convenient service for Spark jobs for inter-datacenter transfer optimization. In this section, we present the architecture overview of our prototype and elaborate the implementation details of major components.

### 6.4.1 Architecture Overview

Fig. 6.2 gives an architecture overview of the software-defined inter-datacenter overlay network, based on which our transfer optimization service is implemented. The testbed consists of a centralized controller and several proxy nodes distributed in each datacenter, as highlighted with the shaded boxes in the figure.

Adhering to the principle of Software Defined Networking (SDN), the control plane and data plane in our testbed are fully separated. The centralized controller has the

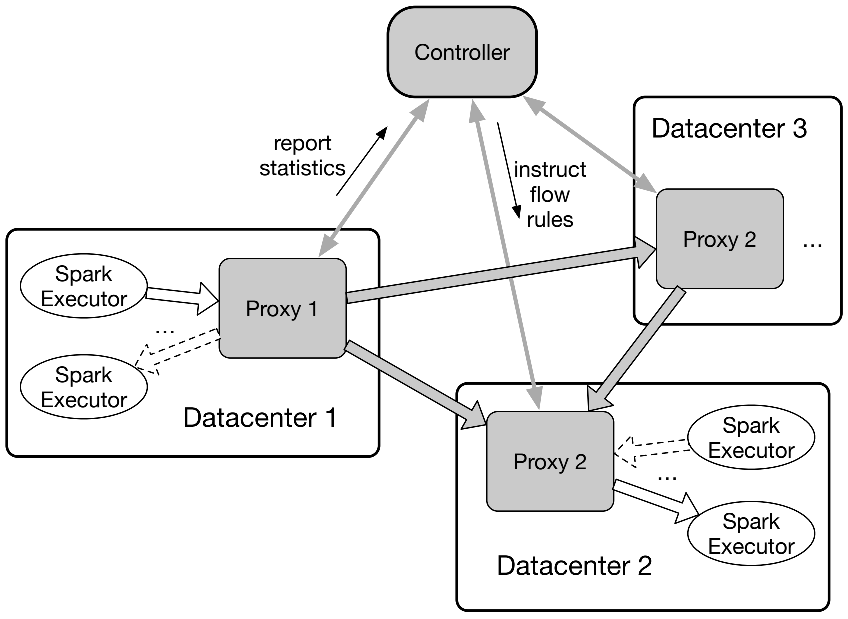


Figure 6.2: Overview of our inter-datacenter overlay testbed.

global view of the network states, reported from the proxy nodes in each datacenter. Our optimal multi-path routing and scheduling strategy is implemented as a controller module, which informs the decisions to proxy nodes for enforcement.

The data plane consists of the proxy nodes in each datacenter, which are responsible for interacting with Spark jobs and enforcing the decisions instructed by the controller. To be more specific, each proxy is designed and implemented as a high-performance switch at the application layer, which aggregates outgoing flows destined to the same datacenter and forwards them along multiple paths at specific rates according to the controller decisions. The data transport module in the Spark framework is modified to send and receive all inter-datacenter traffic through the proxy nodes via a simple API, to be elaborated later.

## 6.4.2 Implementation of Multi-path Routing and Scheduling

Our algorithm of multi-path routing and scheduling is implemented in Python, as a pluggable module in the centralized controller. As the input of our algorithm, the inter-datacenter link bandwidth is readily available in the controller, since it can be measured and reported by the proxy nodes periodically. The set of paths for each pair of datacenters is also available, which can be easily obtained by an exhaustive search given a small number of datacenters. To obtain the coflow set and the size of each flow, we modify the `TaskScheduler` module in Spark to report details of network flows in a shuffle. `TaskScheduler` is a component in Spark which decides the placement of all the tasks in a stage. Once the placement decision has been made, we can have the full knowledge (source, destination and the data size) of all the flows in the corresponding shuffle. Our modified `TaskScheduler` can aggregate such coflow information and reports it to the controller. Upon receiving the report, the controller will trigger the execution of our algorithm.

With all the input available, our algorithm will be launched to construct linear programming problems using sparse matrices provided in the `cvxopt` package. The LP problems are then solved with the commercial LP solver in `Mosek` [17]), which is more efficient than the built-in LP solver in `cvxopt`. The calculated routing and scheduling decision for each flow will be conveyed to all the proxy nodes along its assigned path, where the forwarding rules, in the form of `(flowId, nextHops, rates)`, will be installed and enforced in the data plane. Specifically, `nextHops` is an array which specifies the next-hop proxy nodes to relay the traffic, if the flow is to be split along multiple paths. The other array `rates` specifies the assigned bandwidth on the link to the corresponding next hop.

Upon receiving the forwarding rules, the proxy node will enforce the routing and scheduling decisions, by sending flow fragments to the desired next hops in a round-robin manner. Given a next-hop proxy node, it will first re-fragmentize the flow to a fragment, with the size proportional to the assigned bandwidth on the next-hop link. Then, it will relay the flow fragment to the particular next hop, before proceeding to the next `nextHop` in the array. Since fragments from different flows are also served by round-robin on each link, rate assignment is automatically enforced. Note that the flow re-fragmentization is a feature provided by the proxy nodes at the application-layer, which is implemented with little overhead because of the use of zero-copy buffers and smart pointers.

### 6.4.3 Integration with Spark

To integrate our transport service with Apache Spark, we need to redirect all its inter-datacenter traffic through our proxy nodes, rather than directly initiating connections between executors. Hence, we have modified the `DataTransferService` module in Spark, which is responsible for data block transfers between executors.

With our modification, if a data block is destined to an executor in another datacenter, it will be forwarded to the local proxy by calling a simple function `publishData()`. For a data block destined to a local executor, the default way will be used to initiate a TCP connection between the executors. In a similar vein, to receive data destined to itself through our transport service, a Spark executor needs to call the function `subscribe()`. Both of these calls are implemented as streaming RPC calls, which are supported by the `gRPC`[1] framework using HTTP/2-based transport at the application layer. Our modification is transparent to Spark users, since no changes need to be made to the data analytic jobs.

## 6.5 Performance Evaluation with Real-World Experiments

In this section, we present our evaluation results with a comprehensive set of real-world experiments and extensive simulations, to demonstrate the effectiveness of our multi-path routing and scheduling strategy in optimizing inter-datacenter network transfers and improving job completion times.

**Experiment Setup.** We deploy our optimal transfer service for a Spark cluster, which spans across 5 datacenters on Google's Cloud Compute Engine, with a total of 17

[1]https://grpc.io

Virtual Machine (VM) instances. Specifically, 8 VM instances are evenly distributed in Taiwan and Belgium datacenters, 5 VMs are used in N. Carolina, 3 VMs in Oregon and 1 VM in Tokyo. Each VM instance has 2 vCPUs, 13 GB of memory, and a 20 GB SSD of disk storage. One of the VMs in N. Carolina serves as the Spark master, and the rest of the VMs are the Spark workers, running Apache Spark 2.1.0, the latest release as of January, 2017. The controller is deployed on the same VM as the Spark master, in order to minimize their communication overhead. Two proxy nodes are co-located with Spark workers (or executors) in each datacenter.

**Methodology.** In order to have a fair comparison, we modify the `TaskScheduler` module in Spark, to eliminate the randomness in task placement decisions. In other words, each task in a given workload will be placed on a specific executor across different runs, so that the inter-datacenter traffic generated will be the same. We have evaluated the effectiveness of our strategy (Algorithm 5) in optimizing performance for a variety of data analytic workloads, compared with the direct transfer strategy which only uses direct paths. To the best of our knowledge, our work is the first network optimization for the geo-distributed jobs. The most related works [15, 9] are designed for coflows within a single datacenter. We have not implemented their strategies for fair comparison at the current stage, which will be left as our future work, but a simulation-based comparison has been provided instead to be presented later. Multi-dimensional metrics are measured, including the job completion time, stage completion time and shuffle completion time, to offer a comprehensive comparison.

**Machine Learning Workloads.** We use three representative machine learning workloads from Spark-Perf Benchmark[2], which is the official Spark performance test suite created by Databricks[3]. These workloads are:

- *PCA:* Principle Component Analysis.

[2]`https://github.com/databricks/spark-perf`
[3]`https://databricks.com/`

- *BMM:* Block Matrix Multiplication.
- *Pearson:* Pearson's correlation.

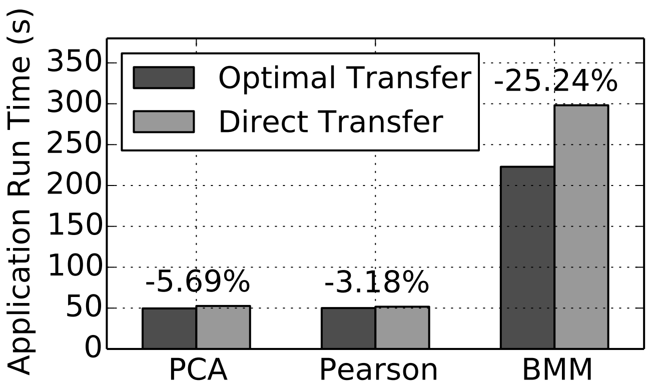


Figure 6.3: Job completion times of three machine learning jobs.

The job completion times (or the application run times, equivalently) achieved by each of the workload with comparing strategies are shown in Fig. 6.3. As expected, our optimal transfer strategy always outperforms the baseline, achieving a 5.69%, 3.18% and 25.24% reduction of completion time for *PCA*, *Pearson* and *BMM* workloads, respectively. It is worth noting that *BMM* enjoys the most significant performance improvement from our strategy, with a 25.24% reduction in its job completion time. The reason is that it is the most network-intensive workload, with a huge shuffle sending more than 40 GB of data, and the inter-datacenter flows it generates are skew in their sizes.

To allow an in-depth analysis of the reason for such a job-level improvement, we present the completion times for each stage of each workload, achieved with the two strategies, respectively, for comparison. The completion time of each stage is decomposed into the network transfer time (shuffle time) and the computation time, as illustrated in Fig. 6.4, Fig. 6.5 and Fig. 6.6.

As shown in Fig. 6.4, a *PCA* job has 4 stages, represented as 0 to 3 along the $x$-axis. Stage 0 and 2 are computation stages, while stage 1 and 3 consist of shuffles. As the job completion time of *PCA* is dominated by the computation time in stage 0 and

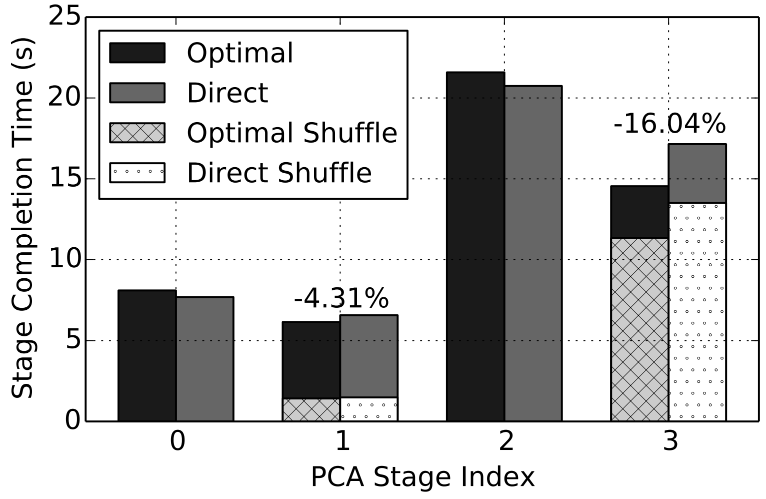


Figure 6.4: Stage completion time of Principal Component Analysis.

2, the job-level performance does not achieve a significant improvement, as reflected in Fig. 6.3. However, with respect to the performance of shuffles, our strategy is effective in reducing the shuffle completion times, by 4.31% for stage 1 and 16.04% for stage 3, respectively. The similar analysis applies to the *Pearson* job illustrated in Fig. 6.5, which is also computation-intensive. The shuffles in both stage 3 and 5 have been accelerated by our strategy, with an improvement of 16.07% and 20.57%, respectively.

In contrast, the *BMM* job is network-intensive, so that its job-level performance is significantly influenced by the performance of its shuffles. As demonstrated in Fig. 6.6, stage 6, which involves a shuffle, dominates the job completion time. Our optimal strategy achieves a 29.98% reduction in the shuffle completion time, which directly contributes to the significant improvement of job-level performance shown in Fig. 6.3.

To offer a more in-depth examination on the shuffle performance of the *BMM* job, we present the CDF of the shuffle read time for each reduce task, which is the time it takes for a reduce task to finish receiving all its required data, in Fig. 6.7. Indeed, the set of flows initiated by each reduce task is naturally treated as a coflow, as the shuffle read time is determined by its slowest flow. With multiple reducers in a shuffle, the inter-datacenter links are shared by multiple such coflows. Recalling Algorithm 5,

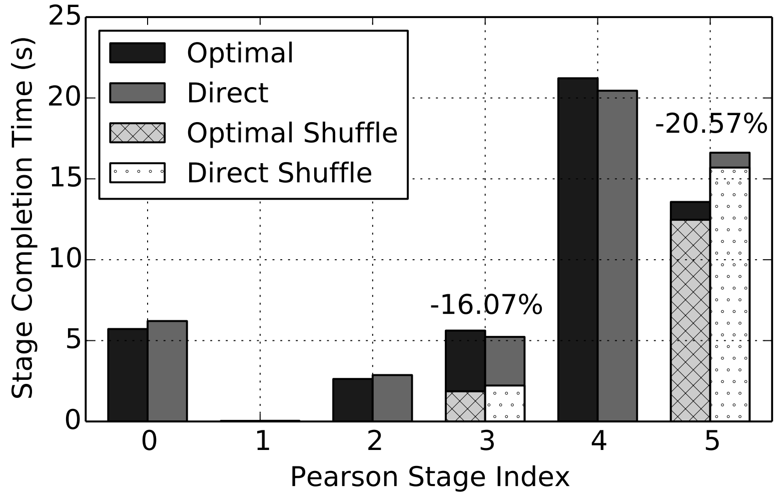


Figure 6.5: Stage completion time of Pearson's correlation.

our strategy tries to minimize the worst coflow completion time repeatedly, until all the coflows have achieve their best possible performance. As clearly shown, our optimal strategy successfully reduces the slowest shuffle read time, from 225 s to 160 s. This results in a significant improvement of the shuffle completion time, which is determined by the slowest shuffle read time intuitively. Moreover, the shuffle read times are more evenly distributed, since our strategy always seeks to optimize for the slowest one. In summary, with our optimal routing and scheduling, the inter-datacenter link bandwidth is more efficiently utilized, so that the shuffle phase is accelerated.

**Sort.** Furthermore, we run the `Sort` application from the HiBench benchmark suite [71], which has only a map stage to sort input data locally and a reduce stage to sort after a heavy shuffle. We generate 2.73 GB of raw input data, which has a skew distribution across the five datacenters. Fig. 6.8 illustrates the reduce completion times of the sort job achieved by the comparing strategies, respectively. Each of the reduce completion times is further decomposed into the shuffle read time and the task execution time, to offer a more comprehensive comparison. It is easy to observe that our strategy effectively improves the shuffle read time, which contributes to the acceleration of the reduce stage. We further present the CDFs of the shuffle read times achieved by the comparing strategies,

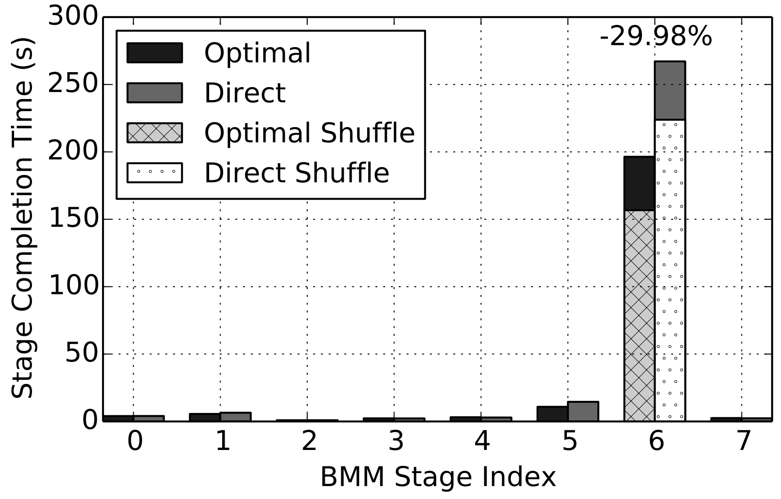


Figure 6.6: Stage completion time of Block Matrix Multiplication.

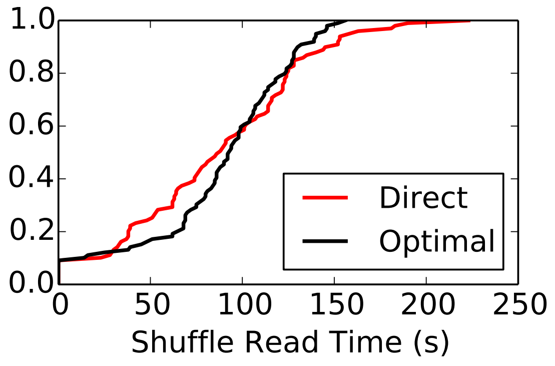


Figure 6.7: CDF of the shuffle read time of the *BMM* job.

respectively, in Fig. 6.9. Compared with the CDF of the baseline that exhibits a long tail, the shuffle read times in our strategy are more balanced, so that the slowest one is accelerated by 2 s.

**Simulation-Based Comparison.** As aforementioned, we further present our comparison with *Rapier* [9], the state-of-the-art for intra-datacenter coflow scheduling and routing, over 100 runs of simulation. In each run, we simulate 4 coflows in a 4-datacenter scenario. Each coflow has 4 inter-datacenter flows, the sizes of which range from 200 to 500 MB. The completion times of the slowest coflows achieved in each of the strategies, respectively, are measured, based on which we obtain our performance improvement ratio

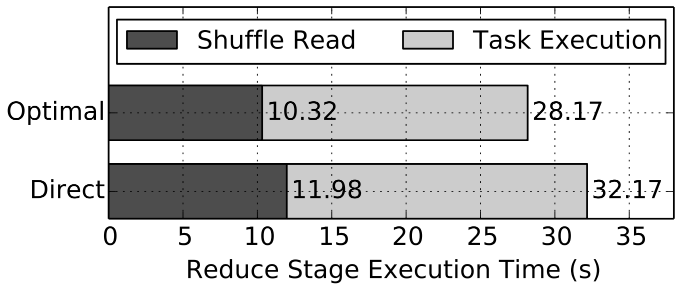


Figure 6.8: The execution time of the reduce stage of the *Sort* job.

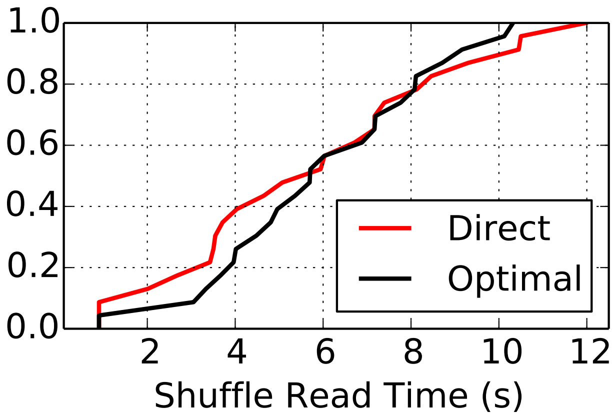


Figure 6.9: CDF of the shuffle read time of the *Sort* job.

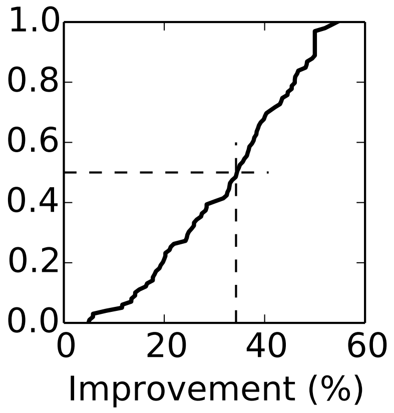


Figure 6.10: CDF of our performance improvement ratio over *Rapier*.

over *Rapier* as the reduction percentage of the worst coflow completion time. The CDF of the performance improvement ratio is presented in Fig. 6.10. Clearly, our strategy outperforms *Rapier* in accelerating the slowest coflow, with an improvement ratio from 5% to 55%, depending on the skewness of the inter-datacenter traffic size. Over 50% of the 100 runs achieve more than 35% of performance improvement.

## 6.6 Summary

In this chapter, we have conducted a thorough study on optimizing the inter-datacenter transfers from multiple sharing data analytic jobs whose tasks are geographically distributed across multiple datacenters. Taking the fact into consideration that multiple paths are available for each inter-datacenter flow, we theoretically investigate the multi-path routing problems for inter-datacenter flows from a single job, and those from multiple concurrent jobs, respectively. For the single job scenario, we formulate the multi-path routing problem as a linear programming to be efficiently solved, which minimizes the completion time of the slowest flow. When multiple jobs are considered, our objective is to optimize their shuffle completion times with max-min fairness, which means that the slowest shuffle achieves its fastest completion time and the same for the next slowest one. To achieve this objective, we have designed an algorithm to iteratively optimize the shuffle completion times by solving an updated version of an LP problem. Last but not the least, we have implemented our performance-optimal routing and scheduling strategy, and provided convenient APIs for Spark users to use our service for network performance optimization. Our real-world experiments over Google Cloud with various workloads demonstrated convincing evidence on the effectiveness and advantages of our new strategy.

# Chapter 7

# Concluding Remarks

## 7.1 Conclusion

In this dissertation, we have investigated the following research problems centering on the theme of improving performance of data analytics jobs, involving flow scheduling, job scheduling and multi-path routing in datacenter networks.

We first focus on the design and implementation of a new utility optimal scheduler across competing coflows in a shared cluster, which have different degrees of sensitivity to their completion times, modeled by their respective utility functions. Our objective is to provide differential treatment to coflows with different degrees of sensitivity, yet still satisfying max-min fairness across these coflows. Though this objective can be formulated as a lexicographical maximization problem, it is challenging to solve in practice due to its inherent multi-objective and discrete nature. To address this challenge, we first divide the problem into iterative steps of single-objective subproblems; and in each of these steps, we then perform a series of transformations to obtain an equivalent linear programming (LP) problem, which can be efficiently solved in practice. To demonstrate that our solutions are practically feasible, we have implemented it as a real-world coflow scheduler based on the Varys open-source framework to evaluate its effectiveness.

We then focus on the problem of assigning tasks belonging to competing data analytics jobs, whose input data are distributed across geo-distributed datacenters. As tasks from multiple jobs are competing for the computing slots in each datacenter, our objective is to achieve max-min fairness across these jobs, in terms of their job completion times. We formulate this problem as a lexicographical minimization problem. To address the challenges incurred by its multi-objective and discrete nature, we iteratively solve its single-objective subproblems, which can be transformed to equivalent LP problems to be efficiently solved, thanks to their favorable properties. We have designed and implemented our proposed solution as a fair job scheduler based on Apache Spark. With extensive evaluations of our real-world implementation on Amazon EC2 and large-scale simulations, we have shown convincing evidence that max-min fairness has been achieved and the worst job completion time has been significantly improved using our new job scheduler.

We further optimize the inter-datacenter network transfers in communication stages of data analytics jobs, by exploiting the flexibility of multi-path routing for inter-datacenter flows to better utilize inter-datacenter links. We design an optimal multi-path routing and scheduling strategy to achieve the best possible network performance for all concurrent jobs, based on our formulation of an optimization problem that can be transformed into an equivalent LP problem to be efficiently solved. We implement our proposed algorithm in the controller of an application-layer software-defined inter-datacenter overlay testbed, designed to provide transfer optimization service for Spark jobs, and conduct extensive evaluations on Google Cloud.

## 7.2 Future Directions

In the future, I look forward to exploring more research issues related to performance optimization for big data jobs in cloud computing. Beyond the topics of network optimization and resource allocation for data parallel jobs, I hope to expand my research areas into optimization for a broad range of distributed data analytics systems, from the perspectives of both framework optimization and transport acceleration, with more emphasis on system design and implementation. Moreover, due to the sharing nature of resources in cloud computing, a unified resource sharing system among a diverse range of machine learning applications is expected to be designed, to accommodate different performance requirements and fairness constraints across various applications. These research directions are elaborated as follows.

**Distributed System for Machine Learning and Data Analytics in the Wide Area.** In the era of big data where everything is personalized and connected, the amount of data to be processed is increasing faster than Moore's Law. With such volume and velocity, various machine learning systems at large scale are developed to handle the extensive amount of data at high speed. As it is inevitable that frequent exchange of intermediate data exists in these systems, network easily becomes the bottleneck, especially in the wide area deployment, which has become the new trend as many large companies have their data generated and stored across geographically distributed datacenters. It is intuitive that deploying the traditional data analytics systems in the wide area leaves the application performance in the wild. Rather than rearranging the task collocation in the data parallelism system or the adaptive synchronization method in the model parallelism system, I believe we need to design a new system from scratch, with well-planed data ingestion, computing, storage, and synchronization, that is perfectly suitable for the inter-datacenter deployment.

**Network Optimization for General Big Data Jobs.** Distributed data analytics depends heavily on the reliability and the performance of the underlying networks, due mainly to its abundant demand for data exchange among tasks. However, the networks, being best-effort and shared among many, can hardly deliver consistent or predictable Quality of Service, especially in the wide area networks with low bandwidth and high latency. As such, complementary to the framework-level optimization aforementioned, the network-level optimization for transport acceleration also plays a significant role in improving the performance of big data jobs. Extending my past work which orchestrates inter-datacenter flows at the application-layer, I would like to further explore the rich space of performance improvement from the lower layer protocol design and rate allocation, given the availability of experimental testbed.

**General Resource Sharing System for Diverse Applications.** The cloud is a shared infrastructure which is expected to accommodate a broad range of applications with different characteristics. It is important to investigate the fundamental principles in sharing multi-dimensional resources across various types of competing applications. This is envisioned to be complementary to the existing schedulers which only focus on particular types of applications or resources. The challenges lie in the accurate modeling of application requirements. To efficiently utilize resources, we need to understand the relationship between performance gain and allocated resource, which becomes more complicated than the simplified linear relationship in previous efforts. Moreover, the principle of fairness also needs to be revisited when applications have different evaluations of the allocated resources. From the theoretical perspective, I would like to seek for an appropriate notion of fairness for this general context and explore the inherent tradeoff between fairness and efficiency. The ultimate goal is a practical resource sharing system which can be general and adaptive to a dynamic environment.

In summary, I look forward to exploring the theoretical and practical challenges in large-scale distributed systems for big data analytics. Rigorous formulation and theoretical study of real-world problems in this broad area can be conducted for practical system design and implementation, with real impacts.

# Bibliography

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for in-Memory Cluster Computing," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012.

[3] M. Chowdhury, M. Zaharia, J. Ma, M. Jordan, and I. Stoica, "Managing Data Transfers in Computer Clusters with Orchestra," in *Proc. ACM SIGCOMM*, 2011.

[4] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: Sharing the Network in Cloud Computing," in *Proc. ACM SIGCOMM*, 2012.

[5] M. Chowdhury and I. Stoica, "Coflow: A Networking Abstraction for Cluster Applications," in *Proc. ACM SIGCOMM HotNet Workshop*, 2012.

[6] Z. Huang, B. Balasubramanian, M. Wang, T. Lan, M. Chiang, and D. H. Tsang, "Need for Speed: CORA Scheduler for Optimizing Completion-Times in the Cloud," in *Proc. IEEE INFOCOM*, 2015.

[7] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient Coflow Scheduling with Varys," in *Proc. ACM SIGCOMM*, 2014.

[8] Z. Qiu, C. Stein, and Y. Zhong, "Minimizing the Total Weighted Completion Time of Coflows in Datacenter Networks," in *Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2015.

[9] Y. Zhao, K. Chen, W. Bai, Y. Minlan, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang, "RAPIER: Integrating Routing and Scheduling for Coflow-Aware Data Center Networks," in *Proc. IEEE INFOCOM*, 2015.

[10] Microsoft Datacenters. [Online]. Available: https://www.microsoft.com/en-ca/cloud-platform/global-datacenters

[11] Google Datacenter Locations. [Online]. Available: https://www.google.com/about/datacenters/inside/locations/

[12] A. Vulimiri, C. Curino, P. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global Analytics in the Face of Bandwidth and Regulatory Constraints," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

[13] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low Latency Geo-Distributed Data Analytics," in *Proc. ACM SIGCOMM*, 2015.

[14] Z. Hu, B. Li, and J. Luo, "Flutter: Scheduling Tasks Closer to Data Across Geo-Distributed Datacenters," in *Proc. IEEE INFOCOM*, 2016.

[15] Y. Li, S. Jiang, H. Tan, and C. Zhang, "Efficient Online Coflow Routing and Scheduling," in *Proc. ACM MobiHoc*, 2016.

[16] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 3rd ed., ser. Algorithms and Combinatorics. Springer, 2006, vol. 21, ch. 5, p. 104.

[17] E. Andersen and K. Andersen, "The MOSEK Interior Point Optimizer for Linear Programming: an Implementation of the Homogeneous Algorithm," in *High Performance Optimization.* Springer, 2000, pp. 197–232.

[18] N. McKeown, "Software-Defined Networking," *INFOCOM keynote talk*, 2009.

[19] L. Chen, Y. Feng, B. Li, and B. Li, "Efficient Performance-Centric Bandwidth Allocation with Fairness Tradeoff," *IEEE Trans. on Parallel and Distributed Systems*, 2018.

[20] L. Chen, W. Cui, B. Li, and B. Li, "Optimizing Coflow Completion Times with Utility Max-Min Fairness," in *Proc. IEEE INFOCOM*, 2016.

[21] L. Chen, S. Liu, B. Li, and B. Li, "Scheduling Jobs across Geo-Distributed Datacenters with Max-Min Fairness," *IEEE Trans. on Network Science and Engineering*, 2018.

[22] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized Task-Aware Scheduling for Data Center Networks," in *Proc. ACM SIGCOMM*, 2014.

[23] H. Susanto, H. Jin, and K. Chen, "Stream: Decentralized Opportunistic Inter-Coflow Scheduling for Datacenter Networks," in *Proc. IEEE International Conference on Network Protocols (ICNP)*, 2016.

[24] M. Chowdhury and I. Stoica, "Efficient Coflow Scheduling Without Prior Knowledge," in *Proc. ACM SIGCOMM*, 2015.

[25] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark," in *Proc. ACM SIGCOMM*, 2016.

[26] A. Vulimiri, C. Curino, P. Godfrey, K. Karanasos, and G. Varghese, “WANalytics: Analytics for A Geo-Distributed Data-Intensive World,” in *Proc. Conference on Innovative Data Systems Research (CIDR)*, 2015.

[27] K. Kloudas, M. Mamede, N. Preguiça, and R. Rodrigues, “Pixida: Optimizing Data Parallel Jobs in Wide-Area Data Analytics,” *VLDB Endowment*, vol. 9, no. 2, pp. 72–83, 2015.

[28] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, “Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds,” in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[29] R. Viswanathan, G. Ananthanarayanan, and A. Akella, “Clarinet: Wan-Aware Optimization for Analytics Queries,” in *Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[30] C. Hung, L. Golubchik, and M. Yu, “Scheduling Jobs Across Geo-Distributed Datacenters,” in *Proc. ACM Symposium on Cloud Computing (SoCC)*, 2015.

[31] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling,” in *Proc. ACM European Conference on Computer Systems*, 2010, pp. 265–278.

[32] B. Hindman, A. Konwinski, M. Zaharia, and et al., “Mesos: A Platform for Fine-Grained Resource Sharing in The Data Center,” in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

[33] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu, “Hopper: Decentralized Speculation-aware Cluster Scheduling at Scale,” in *Proc. ACM SIGCOMM*, 2015.

[34] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM CCR*, vol. 38, no. 2, pp. 69–74, 2008.

[35] "Open Network Foundation Official Website," https://www.opennetworking.org/.

[36] "OpenFlow white paper," https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf, [Online; accessed 6-May-2015].

[37] S. Jain, A. Kumar, S. Mandal *et al.*, "B4: Experience With a Globally-Deployed Software Defined WAN," in *Proc. ACM SIGCOMM*, 2013.

[38] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving High Utilization with Software-Driven WAN," in *Proc. ACM SIGCOMM*, 2013.

[39] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. Rexford, "Optimizing Bulk Transfers with Software-Defined Optical WAN," in *Proc. ACM SIGCOMM*, 2016.

[40] T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese, "NetShare: Virtualizing Data Center Networks across Services," UCSD-CSE, Tech. Rep. CS2010-0957, May 2010.

[41] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the Data Center Network," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

[42] J. Guo, F. Liu, D. Zeng, J. Lui, and H. Jin, "A Cooperative Game Based Allocation for Sharing Data Center Networks," in *Proc. IEEE INFOCOM*, 2013.

[43] J. Guo, F. Liu, H. Tang, Y. Lian, H. Jin, and J. C. Lui, “Falloc: Fair Network Bandwidth Allocation in IaaS Datacenters via A Bargaining Game Approach,” in *Proc. IEEE ICNP*, 2013.

[44] J. Guo, F. Liu, J. Lui, and H. Jin, “Fair Network Bandwidth Allocation in IaaS Datacenters via A Cooperative Game Approach,” *IEEE/ACM Trans. on Networking*, vol. 24, no. 2, pp. 873–886, 2016.

[45] V. Jeyakumar, M. Alizadeh, D. Mazieres, B. Prabhakar, C. Kim, and A. Greenberg, “EyeQ: Practical Network Performance Isolation at the Edge,” in *Proc. USENIX Networked Systems Design and Implementation (NSDI)*, 2013.

[46] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. OâĂŹShea, “Chatty Tenants and the Cloud Network Sharing Problem,” in *Proc. USENIX Networked Systems Design and Implementation (NSDI)*, 2013.

[47] L. Popa, P. Yalagandula, S. Banerjee, and J. Mogul, “ElasticSwitch: Practical Work-Conserving Bandwidth Guarantees for Cloud Computing,” in *Proc. ACM SIGCOMM*, 2013.

[48] G. Kumar, M. Chowdhury, S. Ratnasamy, and I. Stoica, “A Case for Performance-Centric Network Allocation,” in *Proc. USENIX HotCloud Workshop*, 2012.

[49] L. Chen, Y. Feng, B. Li, and B. Li, “Towards Performance-Centric Fairness in Datacenter Networks,” in *Proc. IEEE INFOCOM*, 2014.

[50] L. Chen, B. Li, and B. Li, “Barrier-Aware Max-Min Fair Bandwidth Sharing and Path Selection in Datacenter Networks,” in *Proc. IEEE International Conference on Cloud Engineering (IC2E)*, 2016.

[51] ——, “Surviving Failures with Performance-Centric Bandwidth Allocation in Private Datacenters,” in *Proc. IEEE International Conference on Cloud Engineering (IC2E)*, 2016.

[52] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan, "Large-Scale Parallel Collaborative Filtering for The Netflix Prize," in *Algorithmic Aspects in Information and Management.* Springer, 2008, pp. 337–348.

[53] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving Datacenter Performance and Robustness with Multipath TCP," in *Proc. ACM SIGCOMM*, 2011.

[54] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *Proc. ACM SIGCOMM*, 2009.

[55] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards Predictable Datacenter Networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 242–253.

[56] M. Welzl, *Network Congestion Control: Managing Internet Traffic.* John Wiley & Sons, 2005.

[57] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge University Press, 2004.

[58] H. Yaïche, R. Mazumdar, and C. Rosenberg, "A Game Theoretic Framework for Bandwidth Allocation and Pricing in Broadband Networks," *IEEE/ACM Trans. Networking*, vol. 8, no. 5, pp. 667–678, 2000.

[59] G. Wang, T. Ng, and A. Shaikh, "Programming Your Network at Run-Time for Big Data Applications," in *Proc. ACM HotSDN Workshop*, 2012.

[60] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory Networking: an API for Application Control of SDNs," in *Proc. ACM SIGCOMM*, 2013.

[61] A. Das, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and C. Yu, "Transparent and Flexible Network Management for Big Data Processing in the Cloud," in *Proc. USENIX HotCloud Workshop*, 2013.

[62] M. Alizadeh, S. Yang, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "Deconstructing Datacenter Packet Transport," in *Proc. of ACM SIGCOMM HotNet Workshop*, 2012.

[63] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric," in *Proc. ACM SIGCOMM*, 2009.

[64] R. Meyer, "A Class of Nonlinear Integer Programs Solvable by A Single Linear Program," *SIAM Journal on Control and Optimization*, vol. 15, no. 6, pp. 935–946, 1977.

[65] Breeze: A Numerical Processing Library for Scala. [Online]. Available: http://www.scalanlp.org

[66] Y. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computing Surveys (CSUR)*, vol. 31, no. 4, pp. 406–471, 1999.

[67] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz, "Selecting the Best VM across Multiple Public Clouds: A Data-Driven Performance Modeling Approach," in *Proc. ACM Symposium on Cloud Computing (SOCC)*, 2017.

[68] Hadoop. [Online]. Available: https://hadoop.apache.org/

[69] CPLEX Optimizer: High-Performance Mathematical Programming Solver. [Online]. Available: https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[70] L. Chen, S. Liu, B. Li, and B. Li, "Scheduling Jobs across Geo-Distributed Datacenters with Max-Min Fairness," in *Proc. IEEE INFOCOM*, 2017.

[71] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis," in *Proc. International Conference on Data Engineering Workshops (ICDEW)*, 2010.